SOUTHWEST RESEARCH INSTITUTE
Post Office Drawer 28510, 6220 Culebra Road
San Antonio, Texas 78228-0510

CONTINUATION OF RESEARCH INTO LANGUAGE CONCEPTS FOR THE MISSION SUPPORT ENVIRONMENT

SOURCE CODE

NASA Grant No. NAG 9-435
SwRI Project No. 05-3531

Prepared by:

Timothy J. Barton
Jeremiah M. Ratner

Prepared for:

NASA
Johnson Space Center
Houston, TX 77058

September 5, 1991

Approved:

Melvin A. Schrader

Melvin A. Schrader, Director
Data Systems Department

```
/*****************************************<----->*****************************************
*
* FILE NAME:    build.c
*
* FILE FUNCTION:
*
*   This file contains the routines which allow source code files to be created and
*   compiled.  The routines contained in this file are invoked when the "install"
*   button is selected on the GCB program.
*
*   The build process assumes that all elements and the COMP exist in the current
*   directory.  The following routines will create source files in the current directory
*   for the COMP or element selected.  The routines will then be passed to the compiler
*   for the generation of object files (in the case of an element) or an executable file
*   (in the case of a COMP).
*
*   The build process places as much code generation as possible on the preprocessor
*   as possible.  As a result, whenever and element is to be installed, an include file
*   is created (using the COMP name as the name) to contain all external variable
*   declarations.  The include file will also contain macro definitions which are used
*   to convert tokens from GCB nomenclature to the target source language.
*
*   Global variables are defined (storage allocated) when the main COMP is installed.
*   When the main COMP is installed, a file called "skeleton_element.o" is included
*   on the command line to provide support for process initiation/termination and
*   matrix operations.
*
*   The GCB "start" and "stop" is implemented through the UNIX fork, exec, and kill
*   functions.  The file "skeleton_element.o" contains the code necessary to spawn and
*   terminate COMPs through the use of the "start" and "stop" statements".  The routines
*   track a COMP ID (i.e. a character string) against the UNIX PID number so that the
*   routines can be manipulated.  The routines will allow a finite number of processes
*   to be "start'ed" -- this number is the same as the number of processes allowed by a
*   single user as determined by the UNIX kernel configuration.
*
*   During the execution of the GCB, the following routines assume that the file
*   "skeleton_element.o" exists.  The file should be compiled in the following manner:
*
*       cc -c skeleton_element.c -I.
*
*   in the home directory of the GCB.  The resultant .o file should be placed in the
*   same directory from which the GCB program is initiated.
*
*   In summary, the following source files are used/manipulated by the build routine:
*
*       skeleton_element.c  - source file for run-time utility routines; this file
*                             contains all process manipulation routines and matrix
*                             functions
*       skeleton_element.o  - object file for run-time utility routines
*       skeleton_element.h  - include file for run-time utility routines, this file is
*                             included in all element source files
*       skeleton_element    - source code which is copied into the beginning of EACH
*                             element source file
*       matrix.h            - include file used in matrix manipulation routines
*       xxxxxxx.h           - include file used to compile an element, the include file
*                             uses the COMP as its name (i.e. xxxxxxx = COMP name)
*       yyyyyyy.c           - source file for an element, the name of the source file is
*                             the same as the element name (i.e. yyyyyyy = element name)
*       yyyyyyy.o           - object file for an element, the name of the object file is
*                             the same as the element name (i.e. yyyyyyy = element name)
*       zzzzzzz.c           - source file for a COMP, the name of the COMP source file
*                             is the same as the COMP name (i.e. zzzzzzz = COMP name)
*       zzzzzzz             - executable file for a COMP, the name of the executable
*                             file is the same as the ".c" name (i.e. zzzzzzz =
*                             executable)
*
*   The focus of the GCB prototype was to develop C source code.  Some hooks exist within
*   this file to create other source languages but some re-work will be necessary to
*   fully support other target languages.
*
*
* FILE MODULES:
*
*   add_perins                  - invoked to add perins in relational expressions
*   build_comp_file()           - function to build a COMP
*   build_element_file()        - function to build an ELEMENT
*   cbr_build()                 - entry point from the button selection
*   compute_label_index()       - compute the index into the symbol array of an entry,
*                                 this is used to determine a GOTO label
*   convert_expr_to_C()         - performs necessary conversions from GCB syntax to C
*                                 syntax for expressions
*   convert_to_C_assignment()   - converts a statement to C assignment syntax
*   create_global_variables()   - creates declarations for global variables
*   create_local_variables()    - creates declarations for local variables
*   create_label()              - creates an in-line label
*   create_proc_header()        - creates the appropriate function header syntax
*   create_procedure_name()     - creates the name of a function
*   create_C_string_type()      - converts an integer type into a character string
*   make_header()               - creates the header of an element file
*   make_goto()                 - creates a goto statement
*   make_if()                   - creates an if statement
*   make_pause()                - creates a pause statement
*   make_print()                - creates a print statement
*   make_set()                  - creates a set statement
*   make_start()                - creates a start statement
*   make_stop()                 - creates a stop statement
*   make_trailer()              - creates the end of a function
*   no_label_required()         - determines if a statement needs a preceeding label
*   traverse_line()             - traverses the lines connecting GCB symbols
*
*****************************************<----->*****************************************/

#include <stdio.h>
#include <string.h>
#include <X11/Intrinsic.h>
#include <X11/Shell.h>
#include <Xm/Xm.h>

#include "gcb.h"
#include "symbol.h"
#include "gcb_parse.h"
#include "widgets.h"
#include "element_file.h"

/*
 *  Local defines.
 */

#define SUCCESS                 0

#define NO_CLOSING              0
#define NEEDS_CLOSING           1

#define MAX_COMMENT_HEADER      20
#define MAX_PROCEDURE_NAME      100
#define MAX_LABEL               128
#define MAX_DATA_LINE           1024
#define MAX_ASSIGN_STATEMENT    256
```

```
#define MAX_DATA_TYPE           16
#define MAX_IF_STATEMENT        256
#define MAX_LINE_LENGTH         256
#define MAX_FUNCTION_SIZE       16
#define MAX_EXPRESSION          256
#define MAX_FILENAME            MAX_NAME + 3
#define MAX_OS_COMMAND          1024

#define SKELETON_ELEMENT        "/skeleton_element"
#define SKELETON_OBJECT         "/skeleton_element.o"

#define SEMI_COLON              ";"
#define NEWLINE                 '\n'
#define BLANK                   ' '

/*
 *  The end statement will always be at label L99999.  The following constant is
 *  utilized whenever a goto to the END statement is encountered.
 */

#define END_LABEL               99999

/*
 *  The following global static variable is set to a pointer value when an end is
 *  encountered.  The end is not posted to the output file until the end of all
 *  postings so that the closing of the procedure will be properly handled.
 */

static Symbol *End_To_Be_Posted;

/*
 *  The following global static variable is used to remember the last symbol processed,
 *  this is useful in determining if a label should be generated.
 */

static Symbol *Last_Symbol_Processed;

/*
 *  The following global static variable is set to the name of the main COMP
 *  so that all references will be utilizing the same name.
 */

static char Comp_Name[ MAX_PROCEDURE_NAME ];

/*
 *  Various external routines.
 */

extern struct symbol_entry *lookup_local_varlist();
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  add_perins()
*
* MODULE FUNCTION:
*
*   Function add_perins will add perins to an IF expression to conform to C precadence
*   rules.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

static void add_perins( source, replace_source, replace_dest )
char *source, *replace_source, *replace_dest;
{
    char temp_data[ 1024 ], *remainder, *balance, temp_string[ 1024 ];

    strcpy( temp_data, source );
    while( ( remainder = strstr( temp_data, replace_source ) ) ) {

        /*
         *  Build the string to be exchanged.  Copy the first part of the string to
         *  the temp buffer with enclosing perins, add the modifier and then include
         *  the balance of the string in perins at the end of the string.
         */

        balance = remainder + 4;
        *remainder = NULL;
        sprintf( temp_string, "( %s )%s( %s )", temp_data, replace_dest, balance );
        strcpy( temp_data, temp_string );
    }

    /*
     *  Copy the string back to the original storage space.
     */

    strcpy( source, temp_data );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  compute_label_index()
*
* MODULE FUNCTION:
*
*   Function compute_label_index will determine the index of the symbol into the
*   global array.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

static int compute_label_index( symbol )

    Symbol *symbol;

{
    /*
     *  Compute the offset of the parameter into the Symbol_Map array.  If the symbol
     *  passed to the routine is the end, pass back the generic end label.
     */

    if ( symbol->symbol_type == END )
        return( END_LABEL );
    else
        return( ( (int)symbol - (int)Symbol_Map ) / sizeof( Symbol ) );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  create_proc_header()
*
* MODULE FUNCTION:
*
*   Function create_proc_header is invoked when a procedure header needs to be printed
*   to the target code output file.  The data to be printed comes from various global
*   variables.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

static void  create_proc_header( output_file )

    FILE *output_file;

{
    register int i;
    char    header[ MAX_COMMENT_HEADER ],
            middle[ MAX_COMMENT_HEADER ],
            trailer[ MAX_COMMENT_HEADER ];

    /*
     *  Based on current language preference, build the appropriate leadin comments.
     */

    switch( TargetLanguage ) {
        case C      : strcpy( header, "/*" );
                      strcpy( middle, " *  " );
                      strcpy( trailer, " */" );
                      break;
        case MOAL   : strcpy( header, "" );
                      strcpy( middle, "" );
                      strcpy( trailer, "" );
                      break;
        case UIL    :
                      break;
        default     : displayer( "\n** No target language specified\n" );
                      break;
    }

    /*
     *  Print the necessary procedure header.
     */

    fprintf( output_file, "%s\n", header );
    fprintf( output_file, "%sPosition Name: %s\n", middle, pPosition );
    fprintf( output_file, "%sAuthor:        %s\n", middle, Author );
    fprintf( output_file, "%sCreation Date: %s\n", middle, CreateDate );
    fprintf( output_file, "%sLast Update:   %s\n", middle, UpdateDate );
    fprintf( output_file, "%s\n\n", trailer );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  convert_expr_to_C()
*
* MODULE FUNCTION:
*
*   Function convert_expr_to_C will convert the specified expression to a C expresion
*   from the format entered to the GCB.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

static void convert_expr_to_C( expression )

    char *expression;

{
    char    *prev,
            *ptr,
            *newline;
    int     i,
            len;

    /*
     *  Remove all newlines from the source string.
     */

    while ( newline = strchr( expression, NEWLINE ) )
        *newline = BLANK;

    /*
     *  If the logical operator "and" and "or" appear in the string, then add the
     *  necessary perinthesis to implement C precadence.
     */

    if ( strstr( expression, " or " ) || strstr( expression, " and " ) ) {
        add_perins( expression, " or ", " || " );
        add_perins( expression, " and ", " && " );
    }

    /*
     *  Replace "=" with "==" for C logical expressions.
     */

    ptr = expression;
    while ( *ptr != NULL )
        {

        /*
         *  Locate an equals sign - IFs can not contain assignments so we can
         *  change all = to ==.
         */

        if ( *ptr == '=' )
            {
            prev = ptr-1;

            /*
             *  Check for "<=" and ">=".
             */

            if ((*prev != '>') && (*prev != '<') && (*prev != '!'))
                {
                ptr++;
                len = strlen( ptr );

                /*
                 *  Move all the characters one place to the right so we will have room
                 *  to add the extra =.
                 */

                for ( i=len; i>=0; i-- )    /* move the NULL also */
                    ptr[i+1] = ptr[i];
                *ptr = '=';
                }
            }
        ptr++;
        }

    /*
     *  If the inequality operator of GCB is in the string, replace it with the C
     *  inequality operator.
     */

    while ( strstr( expression, "<>" ) )
        strncpy( strstr( expression, "<>" ), "!=", 2 );
}
```

```
/*************************************<----->*************************************
*
* MODULE NAME:  no_label_required
*
* MODULE FUNCTION:
*
*   Function no_label_required will determinre if a label needs to be generated for
*   the current symbol being processed.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*************************************<----->*************************************/

static int no_label_required( symbol )

    Symbol *symbol;

{
    register int line_count;
    Symbol *temp_symbol;
    LineList *line;

    /*
     *  Count the number of lines pointing to the symbol.
     */

    line_count = 1;
    line = symbol->from;
    while ( line->next ) {
        line_count++;
        line = (LineList *)line->next;
    }

    /*
     *  If more than one line was coming in, the label must be generated.
     */

    if ( line_count > 1 )
        return( 0 );

    /*
     *  If the previous symbol is an IF statement, a label needs to be
     *  drawn based on how code is generated.
     */

    if ( ((Symbol *)(symbol->from->line->from))->symbol_type == IF )
        return( 0 );

    /*
     *  At this point, only one line is coming to the symbol, if the previously
     *  processed symbol is the same as where the line came from, the label does
     *  not have to be generated.
     */

    if ( (Symbol *)symbol->from->line->from == Last_Symbol_Processed )
        return( 1 );

    /*
     *  Issue a code to indicate that a label needs to be generated.
     */

    return( 0 );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  create_label()
*
* MODULE FUNCTION:
*
*   Function create_label is called if a symbol has a line pointing to the symnbol that
*   may require a goto in the code.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

static int create_label( symbol, output_file )

    Symbol  *symbol;
    FILE    *output_file;

{
    /*
     *  If no lines are drawn to the symbol, no label is required so simply return.
     */

    if ( symbol->from == NULL )
        return( 0 );

    /*
     *  If the symbol only has a single line to the symbol and the previously
     *  processed symbol is the one that points to the current symbol, no label
     *  is necessary because they code will naturally fall through.
     */

    if ( no_label_required( symbol ) )
        return( 0 );

    /*
     *  Based on current language preference, build the appropriate label.
     */

    switch( TargetLanguage ) {
        case C      : fprintf( output_file,"\nL%05d:\n", compute_label_index( symbol ));
                      if ( symbol->symbol_type == END )
                          fprintf( output_file, "\t;\n" );
                      break;
        case MOAL   : fprintf( output_file,"\nL%05d\n", compute_label_index( symbol ) );
                      break;
        case UIL    :
                      break;
        default     : displayer( "** No target language specified\n" );
                      return( ERR );
                      break;
    }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  create_procedure_name()
*
* MODULE FUNCTION:
*
*   Create a procedure name by removing any file extension and only use the name after
*   any slashes.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

static void create_procedure_name( source )

    char *source;

{
    register int i, index;

    /*
     *  Copy the name over to the new variable.
     */

    strcpy( Comp_Name, source );

    /*
     *  Remove any trailing extension.
     */

    for ( i = strlen( Comp_Name ) - 1; i >= 0; i-- )
        if ( Comp_Name[ i ] == '.' ) {
            Comp_Name[ i ] = NULL;
            break;
        }

    /*
     *  If any slashes exist in the name, find the last one and copy eliminate any data
     *  before the slash.
     */

    index = -1;
    for ( i = 0; i < strlen( Comp_Name ); i++ )
        if ( Comp_Name[ i ] == '/' )
            index = i;

    if ( index != -1 )
        strcpy( Comp_Name, &Comp_Name[ index + 1 ] );

    /*
     *  If the resulting name has no data left, return a generic name.
     */

    if ( Comp_Name[ 0 ] == NULL )
        strcpy( Comp_Name, "generic_procedure_name" );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  create_C_string_type()
*
* MODULE FUNCTION:
*
*   Function create_C_string_type is invoked to convert an internal integer data type
*   representation to a C named data type (the conversion is used for variable
*   declaration).
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

static void create_C_string_type( data_type, type_string )

    int     data_type;
    char    *type_string;

{
        switch( data_type ) {
            case INTEGER  : strcpy( type_string, "int" );
                            break;
            case UNSIGNED : strcpy( type_string, "unsigned" );
                            break;
            case FLOAT    : strcpy( type_string, "float" );
                            break;
            case CHAR     : strcpy( type_string, "char" );
                            break;
            case DOUBLE   : strcpy( type_string, "double" );
                            break;
            case SHORT    : strcpy( type_string, "short" );
                            break;
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  create_local_variables()
*
* MODULE FUNCTION:
*
*   The function will declare all local variables contained in the symbol table.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

static void create_local_variables( output_file )

    FILE *output_file;

{
    char data_type[ MAX_DATA_TYPE ];
    struct symbol_entry *local_vars;

    /*
     *  Find the list of local variables for the current procedure (element).
     */

    if ( ( local_vars = lookup_local_varlist( ElementFile ) ) == NULL )
        return;

    /*
     *  For each entry on the local variable list, create a variable
     *  decleration.
     */

    while ( local_vars ) {

        /*
         *  Based on the data type, create a variable data type variable.
         */

        create_C_string_type( ( local_vars->se_type & SYMBOL_DATA_TYPE ), data_type );

        /*
         *  Write the variable declaration to the output file (if the variable is a
         *  matrix then allocate the appropriate space).  If the variable is of type
         *  CHAR, then simply declare a pointer to char.
         */

        if ( local_vars->se_type & CHAR )
            fprintf( output_file, "\t%s *%s;\n",
                                  data_type,
                                  local_vars->se_symbol );
        else if ( local_vars->se_type & MATRIX ) {

                /*
                 *  Declare the variable as appropriate based on the number of declared
                 *  dimensions.  At least one dimension will be specified, if additional
                 *  are required, the appropriate code will be generated.
                 */

                fprintf( output_file, "\t%s %s[ %d ]",
```

```
                                data_type,
                                local_vars->se_symbol,
                                local_vars->se_subs[ SUB1 ] );
                    switch( local_vars->se_num_dimensions ) {
                        case 2  : fprintf( output_file, "[ %d ];\n",
                                                local_vars->se_subs[ SUB2 ] );
                                  break;
                        case 3  : fprintf( output_file, "[ %d ][ %d ];\n",
                                                local_vars->se_subs[ SUB2 ],
                                                local_vars->se_subs[ SUB3 ] );
                                  break;
                        case 4  : fprintf( output_file, "[ %d ][ %d ][ %d ];\n",
                                                local_vars->se_subs[ SUB2 ],
                                                local_vars->se_subs[ SUB3 ],
                                                local_vars->se_subs[ SUB4 ] );
                                  break;
                        default : break;
                    }
                }
                else
                    fprintf( output_file, "\t%s %s;\n", data_type, local_vars->se_symbol );

            /*
             *  Goto the next local variable.
             */

            local_vars = local_vars->se_next;
        }
    }
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  make_header()
*
* MODULE FUNCTION:
*
*   Function make_header will post the necessary characters to the target output file
*   to create a C function header.  The GCB does not allow formal parameters to functions
*   so the syntax is quite simple.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

static void make_header( symbol, output_file )

    Symbol  *symbol;
    FILE    *output_file;

{
    /*
     *  Create a name based on the element file name.
     */

    create_procedure_name( ElementFile );

    /*
     *  Based on current language preference, build the appropriate header.
     */

    switch( TargetLanguage ) {
        case C      : fprintf( output_file, "void %s()\n{\n", Comp_Name );
                      create_local_variables( output_file );
                      fprintf( output_file, "\n" );
                      create_label( symbol, output_file );
                      break;
        case MOAL   :
                      break;
        case UIL    :
                      break;
        default     : displayer( "** No target language specified\n" );
                      break;
    }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  make_trailer()
*
* MODULE FUNCTION:
*
*   Function make_trailer will post the necessary characters to the target output file
*   to create a C function trailer.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/


static void make_trailer( output_file )

    FILE *output_file;

{
    /*
     *  Create a label before the end of the function.  This is done because the end
     *  will almost always have a GOTO somewhere in the source file.  This will
     *  sometimes result in two consecutive lines of the C source file to be a "goto"
     *  and then a "label".  These statements will be removed by a good optimizing
     *  compiler. Embedding the logic to remove the extra goto/label into these
     *  routines would needlessly complicate the routines.
     */

    create_label( End_To_Be_Posted, output_file );

    /*
     *  Based on current language preference, build the appropriate trailer.
     */

    switch( TargetLanguage ) {
        case C      : fprintf( output_file, "}\n" );
                      break;
        case MOAL   :
                      break;
        case UIL    :
                      break;
        default     : displayer( "** No target language specified\n" );
                      break;
    }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  make_if()
*
* MODULE FUNCTION:
*
*   Function make_if will create the code for an IF statement.  It is assumed that the
*   logical expression contained in the GCB symbol is syntactically correct.  Note that
*   if's are generated with goto's for both the true and false case; attempting to put
*   source code inline after the if's is not possible given the structuring capabilities
*   of the GCB user interface.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                                 Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

static int make_if( symbol, output_file )

    Symbol  *symbol;
    FILE    *output_file;

{
    register int true, false;
    char expression[ MAX_EXPRESSION ];

    /*
     *  Set the global variable SetSym to indicate that we are parsing a Set symbol;
     *  thus the parser will know what kind of expression to expect.
     */

    SetSym = 0;

    /*
     *  Compute the indecies for the true and false labels.  These indecies will be used
     *  to generate the appropriate goto's in the code generation phase below.
     */

    if ( symbol->Sym.IfSym.true_line )
        true = compute_label_index( symbol->Sym.IfSym.true_line->to );
    else
        true = -1;

    if ( symbol->Sym.IfSym.false_line )
        false = compute_label_index( symbol->Sym.IfSym.false_line->to );
    else
        false = -1;

    /*
     *  Copy the expression to local memory.
     */

    strcpy( expression, symbol->Sym.IfSym.comp_expr );

    /*
     *  Based on current language preference, build the appropriate conditional.  If the
     *  tagrget language is C the expression will be encased in perenthesis so that the
     *  statement will be syntactically correct.
     */
```

```
    switch( TargetLanguage ) {
        case C      : convert_expr_to_C( expression );
                      if ( ( false > -1 ) && ( true > -1 ) )
                        fprintf( output_file,
                               "\tif ( %s )\n\t\tgoto L%05d;\n\telse\n\t\tgoto L%05d;\n"

                               expression, true, false );
                      else if ( ( false == -1 ) && ( true > -1 ) )
                        fprintf( output_file,
                               "\tif ( %s )\n\t\tgoto L%05d;\n",
                               expression, true );
                      else if ( ( false > -1 ) && ( true == -1 ) )
                        fprintf( output_file,
                               "\tif ( %s )\n\t\t;\n\telse\n\t\tgoto L%05d\n",
                               expression, false );
                      else
                        fprintf( output_file,
                               "\tif ( %s )\n\t\t;\n\telse\n\t\t;\n",
                               expression );
                      break;
        case MOAL   : if ( ( false > -1 ) && ( true > -1 ) )
                        fprintf( output_file,
                               "if %s then\n\tgoto L%05d\nelse\n\tgoto L%05d\nendif\n",
                               symbol->Sym.IfSym.comp_expr, true, false );
                      else if ( ( false == -1 ) && ( true > -1 ) )
                        fprintf( output_file,
                               "if %s then\n\tgoto L%05d\nendif\n",
                               symbol->Sym.IfSym.comp_expr, true );
                      else if ( ( false > -1 ) && ( true == -1 ) )
                        fprintf( output_file,
                               "if %s then\nelse\n\t\tgoto L%05d\nendif\n",
                               symbol->Sym.IfSym.comp_expr, false );
                      else
                        fprintf( output_file,
                               "if %s then\nelse\nendif\n",
                               symbol->Sym.IfSym.comp_expr );
                      break;
        case UIL    :
                      break;
        default     : displayer( "** No target language specified\n" );
                      return( ERR );
                      break;
    }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  convert_to_C_assignment()
*
* MODULE FUNCTION:
*
*   Function convert_to_C_assignment will convert the supplied character string to a
*   valid C assignment statement.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                                  Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

static void convert_to_C_assignment( statement )

    char *statement;

{
    /*
     *  Concatenate a semi-colon to the end of the line for the statement terminator.
     */

    strcat( statement, SEMI_COLON );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  make_set()
*
* MODULE FUNCTION:
*
*   Function make_set will create the code for a set command.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

static int make_set( symbol, of )

    Symbol  *symbol;
    FILE    *of;

{
    int     operator,
            type_operation;
    char    statement[ MAX_ASSIGN_STATEMENT ],
            *operand,
            *ptr;
    struct matrix_data result, operand_1, operand_2;

    /*
     *  Set the global variable SetSym to indicate that we are parsing a Set symbol;
     *  thus the parser will know what kind of expression to expect.
     */

    SetSym = 1;

    /*
     *  Copy the expression to local stack space to that modifications can be made w/o
     *  effecting the COMP display.
     */

    strcpy( statement, symbol->Sym.IfSym.comp_expr );

    /*
     *  Determine if the expression is MATRIX or SCALER operation.
     */

    if ( ( type_operation = return_matrix_data( statement,
                                                &operator,
                                                &result,
                                                &operand_1,
                                                &operand_2 ) ) == ERR ) {
        displayer( "Syntax error in expression, cannot build a COMP\n" );
        return( ERR );
    }

    /*
     *  Change the ":-" in the expression to a " -".
     */

    if ( (ptr = strchr(statement,':')) == NULL )
        return( ERR );
    else
        *ptr = ' ';

    /*
     *  If the operation is for a MATRIX statement, build the appropriate calls based
     *  on the specified operator; otherwise, perform a scalar build.  Matrix calls
     *  are made to the routines defined in "skeleton_element.o".
     */

    if ( type_operation == SCALAR_OPERATION )

        /*
         *  Perform a scalar build.
         */

        switch( TargetLanguage ) {
            case C      : convert_to_C_assignment( statement );
                          fprintf( of, "\t%s\n", statement );
                          break;
            case MOAL   : fprintf( of, "set %s\n", statement );
                          break;
            case UIL    :
                          break;
            default     : displayer( "** No target language specified\n" );
                          return( ERR );
                          break;
        }
    else
        /*
         *  Perform a MATRIX build.
         */

        switch( operator ) {
          case ADD_OPER   : fprintf( of, "\tmatrix_add( %d, %s, %s, %d, %d, %s );\n",
                                         result.md_attributes & SYMBOL_DATA_TYPE,
                                         operand_1.md_name,
                                         operand_2.md_name,
                                         result.md_subs[ SUB1 ],
                                         result.md_subs[ SUB2 ],
                                         result.md_name );
                            break;
          case MINUS_OPER : fprintf( of, "\tmatrix_sub( %d, %s, %s, %d, %d, %s );\n",
                                         result.md_attributes & SYMBOL_DATA_TYPE,
                                         operand_1.md_name,
                                         operand_2.md_name,
                                         result.md_subs[ SUB1 ],
                                         result.md_subs[ SUB2 ],
                                         result.md_name );
                            break;
          case MULT_OPER  : fprintf(of,"\tmatrix_mult( %d, %s, %s, %d, %d, %d, %s );\n",
                                         result.md_attributes & SYMBOL_DATA_TYPE,
                                         operand_1.md_name,
                                         operand_2.md_name,
                                         operand_1.md_subs[ SUB1 ],
                                         operand_1.md_subs[ SUB2 ],
                                         operand_2.md_subs[ SUB1 ],
                                         result.md_name );
                            break;
          case IDENT_OPER : fprintf( of, "\tmatrix_ident( %d, %d, %s );\n",
                                         result.md_attributes & SYMBOL_DATA_TYPE,
                                         result.md_subs[ SUB1 ],
                                         result.md_name );
                            break;
          case INVER_OPER : fprintf( of, "\tmatrix_inverse( %d, %s, %d, %s );\n",
                                         result.md_attributes & SYMBOL_DATA_TYPE,
```

```
                                operand_1.md_name,
                                result.md_subs[ SUB1 ],
                                result.md_name );
                        break;
        case TRANS_OPER : fprintf( of, "\tmatrix_transpose( %d, %s, %d, %d, %s );\n",
                                result.md_attributes & SYMBOL_DATA_TYPE,
                                operand_1.md_name,
                                result.md_subs[ SUB1 ],
                                result.md_subs[ SUB2 ],
                                result.md_name );
                        break;
        case CROSS_OPER : fprintf( of, "\tmatrix_cross( %d, %d, %s, %s, %s );\n",
                                result.md_attributes & SYMBOL_DATA_TYPE,
                                result.md_subs[ SUB1 ],
                                operand_1.md_name,
                                operand_2.md_name,
                                result.md_name );
                        break;
        case DOT_OPER   : fprintf( of, "\tmatrix_dot( %d, %s, %s, %d, &%s );\n",
                                operand_1.md_attributes & SYMBOL_DATA_TYPE,
                                operand_1.md_name,
                                operand_2.md_name,
                                operand_1.md_subs[ SUB1 ],
                                result.md_name );
                        break;
        default         :
                        /*
                         *  A simple assignment, if the attributes of the RHS
                         *  indicate a matrix, then call the appropriate routine,
                         *  otherwise invoke the routine to initialize a matrix
                         *  with a scalar routine.
                         *
                         *  This is only supported for 1D and 2D matrices.
                         */

                        if ( operand_1.md_attributes & MATRIX )
                            if ( operand_1.md_num_dimensions <= 2 )
                                fprintf( of,"\tmatrix_copy( %d, %s, %s, %d, %d );\n"

                                        result.md_attributes & SYMBOL_DATA_TYPE,
                                        result.md_name,
                                        operand_1.md_name,
                                        result.md_subs[ SUB1 ],
                                        result.md_subs[ SUB2 ] );
                            else
                                fprintf( of,
                                   "\tmatrix_copy4( %d, %s, %s, %d, %d, %d, %d );\n"

                                        result.md_attributes & SYMBOL_DATA_TYPE,
                                        result.md_name,
                                        operand_1.md_name,
                                        result.md_subs[ SUB1 ],
                                        result.md_subs[ SUB2 ],
                                        result.md_subs[ SUB3 ],
                                        result.md_subs[ SUB4 ] );
                        else {

                            /*
                             *  Need to generate a pointer in to the expression to
                             *  get the RHS variable.  We assume that the variable
                             *  starts immediately after the "=" sign (therefore add
                             *  one to the value returned by strstr to get to the
                             *  spot in the string.  Cast the init value to double
                             *  so that the matrix_init routine will be properly
                             *  invoked.
                             */

                            if ( ( operand = strstr( statement, "=" ) ) == NULL )
                                return( ERR );

                            /*
                             *  Call the appropriate routine based on how many
                             *  dimensions the result routine has.
                             */

                            if ( result.md_num_dimensions <= 2 )
                                fprintf( of,
                                    "\tmatrix_init( %d, %s, %d, %d, (double)%s );\n",
                                        result.md_attributes & SYMBOL_DATA_TYPE,
                                        result.md_name,
                                        result.md_subs[ SUB1 ],
                                        result.md_subs[ SUB2 ],
                                        ( operand + 1 ) );
                            else
                                fprintf( of,
                                    "\tmatrix_init4( %d, %s, %d, %d, %d, %d, (double)%s )
;\n",
                                        result.md_attributes & SYMBOL_DATA_TYPE,
                                        result.md_name,
                                        result.md_subs[ SUB1 ],
                                        result.md_subs[ SUB2 ],
                                        result.md_subs[ SUB3 ],
                                        result.md_subs[ SUB4 ],
                                        ( operand + 1 ) );
                        }
                        break;
                }

        /*
         *  Return a successful return code.
         */

        return( SUCCESS );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  popup_wait()
*
* MODULE FUNCTION:
*
*   This routine does local processing until the user responds to the popup.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  popup_wait()

{
    XEvent  event;

    ActivePopup++;
    PopupStat[ ActivePopup ] = -1;

    /*
     *  throw out events until the PopupStat array is updated
     */

    while ( PopupStat[ ActivePopup ] < 0 )
        {
        XtAppNextEvent( app_context, &event );
        XtDispatchEvent( &event );
        }
     ActivePopup--;
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  process_popup()
*
* MODULE FUNCTION:
*
*   This routine  manages the user_ack popup and does local processing until
*   the user responds.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  process_popup( widget, wait_flag )

    Widget  widget;
    int     wait_flag;

{
    Dimension x,y;

    XtManageChild( widget );

    if ( wait_flag )
        popup_wait();
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  redraw_sym_type()
*
* MODULE FUNCTION:
*
*   This routine redraws all symbols of specified type.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  redraw_sym_type(type)

    int type;

{
    int j;

    for ( j=0; j<MAX_SYMBOLS; j++ )
        {
        if ( Symbol_Map[j].symbol_type == type )
            {

            /*
             *  set global text variables for draw_symbol routine.
             */

            if ( (type == IF) || (type == SET) )
                {
                sym_text = Symbol_Map[j].Sym.IfSym.logical_expr;
                expr_text = Symbol_Map[j].Sym.IfSym.comp_expr;
                }
            else sym_text = Symbol_Map[j].text;
            draw_symbol( type, Symbol_Map[j].mycanvas, WAgc, Symbol_Map[j].font );
            }
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  redraw_sym_num()
*
* MODULE FUNCTION:
*
*   This routine redraws symbol with specified index into Symbol_Map.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  redraw_sym_num( num, font )

    int      num;
    Font     font;

{
    /*
     *  set global text variables for draw_symbol routine.
     */

    if ( (Symbol_Map[num].symbol_type == IF) ||
        (Symbol_Map[num].symbol_type == SET) )
        {
        sym_text = Symbol_Map[num].Sym.IfSym.logical_expr;
        expr_text = Symbol_Map[num].Sym.IfSym.comp_expr;
        }
    else sym_text = Symbol_Map[num].text;
    draw_symbol( Symbol_Map[num].symbol_type, Symbol_Map[num].mycanvas, WAgc, font );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  make_pause()
*
* MODULE FUNCTION:
*
*   Function make_pause will create the code for a pause command.  The function will map
*   the specified pause time (and units) to the appropriate UNIX system call.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

static void make_pause( symbol, output_file )

    Symbol  *symbol;
    FILE    *output_file;

{
    unsigned seconds;
    char function[ MAX_FUNCTION_SIZE ];

    /*
     *  Parse out the integer argument from the text.  Then determine the units
     *  specified, no units specified will default to seconds.
     */

    sscanf( symbol->text, "%u", &seconds );
    strcpy( function, "sleep" );
    if ( strstr( symbol->text, "min" ) != NULL )
            seconds *= 60;
    else if ( strstr( symbol->text, "ms" ) != NULL ) {
            seconds *= 1000;
            strcpy( function, "usleep" );
    }

    /*
     *  Based on current language preference, build the appropriate pause command.
     */

    switch( TargetLanguage ) {
        case C      : fprintf( output_file, "\t%s( %u );\n", function, seconds );
                      break;
        case MOAL   :
                      break;
        case UIL    :
                      break;
        default     : displayer( "** No target language specified\n" );
                      break;
    }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  make_goto()
*
*
* MODULE FUNCTION:
*
*   Function make_goto will create the source code to invoke a C function (i.e. a goto
*   is simply a function call with no actual parameters).
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

static void make_goto( symbol, output_file )

    Symbol  *symbol;
    FILE    *output_file;

{
    /*
     *  Based on current language preference, build function invocation statement.
     */

    switch( TargetLanguage ) {
        case C      : fprintf( output_file, "\t%s();\n",
                                            symbol->text );
                      break;
        case MOAL   : fprintf( output_file, "start( %sL )\n",
                                            symbol->text );
                      break;
        case UIL    :
                      break;
        default     : displayer( "** No target language specified\n" );
                      break;
    }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  make_start()
*
* MODULE FUNCTION:
*
*   Function make_start will create the code to issue a start command.  The start comman
d
*   is to invoke another previously created COMP.  The make_start function will create a
*   function call to the process manipulation routines defined in skeleton_element.o.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

static void make_start( symbol, output_file )

    Symbol  *symbol;
    FILE    *output_file;

{
    /*
     *  Based on current language preference, build the appropriate call to the process
     *  manipulation routine.
     */

    switch( TargetLanguage ) {
        case C      : fprintf( output_file, "\tstart_process( \"%s\" );\n",
                                            symbol->text );
                      break;
        case MOAL   : fprintf( output_file,
                               "if fork() = 0 then\n\texecl( %s, NULL )\nendif\n",
                                symbol->text );
                      break;
        case UIL    :
                      break;
        default     : displayer( "** No target language specified\n" );
                      break;
    }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  make_stop()
*
* MODULE FUNCTION:
*
*   Function make_stop will create the code to issue a stop command.  The stop command
*   is to terminate a previously initiated COMP.  The make_start function will create a
*   function call to the process manipulation routines defined in skeleton_element.o.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

static void make_stop( symbol, output_file )

    Symbol  *symbol;
    FILE    *output_file;

{
    /*
     *  Based on current language preference, build the call to the process termination
     *  routine.
     */

    switch( TargetLanguage ) {
        case C      : fprintf( output_file, "\tstop_process( \"%s\" );\n",
                                            symbol->text );
                      break;
        case MOAL   : fprintf( output_file, "terminate( process_id )\n" );
                      break;
        case UIL    :
                      break;
        default     : displayer( "** No target language specified\n" );
                      break;
    }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  set_attribs()
*
* MODULE FUNCTION:
*
*   This routine sets the width, height, and resize policy resources of the
*   parameter widget.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  set_attribs( type, widget, data1, data2, data3 )

    Widget  widget;
    int     type;
    caddr_t data1,data2,data3;

{
    Arg args[3];

    /*
     *  set various attributes depending on the type of the widget.
     */

    if ( type == FORM )
        {
        XtSetArg( args[0], XmNwidth,        data1 );
        XtSetArg( args[1], XmNheight,       data2 );
        XtSetArg( args[2], XmNresizePolicy, data3 );
        XtSetValues( widget, args, 3 );
        }
    else
        {
        XtSetArg( args[0], XmNselectionPolicy, data1 );
        XtSetValues( widget, args, 1 );
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  set_position()
*
* MODULE FUNCTION:
*
*   This routine sets the top, bottom, left, and right positions of the parameter.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  set_position( w, t, b, l, r )

    Widget  w;
    int     t, b, l, r;

{
    register int        n = 0;
    Arg                 args[8];

    if ( t != IGNORE )
        {
        XtSetArg( args[n], XmNtopAttachment, XmATTACH_POSITION ); n++;
        XtSetArg( args[n], XmNtopPosition, t ); n++;
        }
    if ( b != IGNORE )
        {
        XtSetArg( args[n], XmNbottomAttachment, XmATTACH_POSITION ); n++;
        XtSetArg( args[n], XmNbottomPosition, b ); n++;
        }
    if ( l != IGNORE )
        {
        XtSetArg( args[n], XmNleftAttachment, XmATTACH_POSITION ); n++;
        XtSetArg( args[n], XmNleftPosition, l ); n++;
        }
    if ( r != IGNORE )
        {
        XtSetArg( args[n], XmNrightAttachment, XmATTACH_POSITION ); n++;
        XtSetArg( args[n], XmNrightPosition, r ); n++;
        }
    XtSetValues( w, args, n);
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  set_user_data()
*
* MODULE FUNCTION:
*
*   This routine sets the XmNuserData resource of the parameter resource.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  set_user_data( widget, data )

    Widget  widget;
    int     data;

{
    Arg args[1];

    XtSetArg( args[0], XmNuserData, data );
    XtSetValues( widget, args, 1 );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  set_widget()
*
* MODULE FUNCTION:
*
*   This routine sets the top, bottom, left, and right widgets of the parameter.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  set_attach_widget( w, t, b, l, r )

    Widget  w;
    Widget  t, b, l, r;

{
    register int        n = 0;
    Arg                 args[8];

    if ( t )
        {
        XtSetArg( args[n], XmNtopAttachment, XmATTACH_WIDGET ); n++;
        XtSetArg( args[n], XmNtopWidget, t ); n++;
        }
    if ( b )
        {
        XtSetArg( args[n], XmNbottomAttachment, XmATTACH_WIDGET ); n++;
        XtSetArg( args[n], XmNbottomWidget, b ); n++;
        }
    if ( l )
        {
        XtSetArg( args[n], XmNleftAttachment, XmATTACH_WIDGET ); n++;
        XtSetArg( args[n], XmNleftWidget, l ); n++;
        }
    if ( r )
        {
        XtSetArg( args[n], XmNrightAttachment, XmATTACH_WIDGET ); n++;
        XtSetArg( args[n], XmNrightWidget, r ); n++;
        }
    XtSetValues( w, args, n );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  make_print()
*
* MODULE FUNCTION:
*
*   Function make_print will create the code to issue a print command.  Currently the
*   prototype GCB does not support the printing of variables, therefore the print
*   routine is quite simple.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

static void make_print( symbol, output_file )

    Symbol  *symbol;
    FILE    *output_file;

{
    /*
     *  Based on current language preference, build the appropriate print statement.
     */

    switch( TargetLanguage ) {
        case C      : fprintf( output_file, "\tprintf( \"%s\\n\" );\n",
                                            symbol->text );
                      break;
        case MOAL   : fprintf( output_file, "print( 0, %s )\n",
                                            symbol->text );
                      break;
        case UIL    :
                      break;
        default     : displayer( "** No target language specified\n" );
                      break;
    }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  traverse_line()
*
* MODULE FUNCTION:
*
*   Function traverse_line will traverse the specified list output the symbol structure t
o
*   the target source file.  The appropriate statement building functions will be invoked
*   as they are encounter during the search.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

static void traverse_line( symbol, output_file )

    Symbol  *symbol;
    FILE    *output_file;

{
    /*
     *  Loop through all the symbols linked together.
     */

    while ( symbol ) {

        if ( symbol->symbol_generated == GENERATED )
            return;

        /*
         *  Mark the symbol as generated.
         */

        symbol->symbol_generated = GENERATED;

        /*
         *  Determine if a label should be made for the symbol.  A label is not created
         *  if the symbol is begin (because the label has to be after the local variable
         *  declarations.
         */

        if ( ( symbol->symbol_type != BEGIN ) && ( symbol->symbol_type != END ) )
            create_label( symbol, output_file );

        /*
         *  Based on what the symbol is, traverse the line after outputing the symbol.
         */

        switch( symbol->symbol_type ) {
            case BEGIN: make_header( symbol, output_file );
                        break;
            case END  :
                        /*
                         *  If an end has not yet been encountered, record the current
                         *  position of the end (for use when ends are later
                         *  encountered).
                         */
```

```
                    if ( End_To_Be_Posted == NULL )
                        End_To_Be_Posted = symbol;

                    /*
                     *  Create a label for the first encountered END to generate
                     *  the goto.
                     */

                    switch( TargetLanguage ) {
                        case C    : fprintf( output_file, "\tgoto L%05d;\n",
                                    compute_label_index( End_To_Be_Posted ) );
                                    break;
                        case MOAL : break;
                        case UIL  : break;
                        default   : displayer("** No target language specified\n");
                                    break;
                    }
                    break;
        case IF   : make_if( symbol, output_file );
                    Last_Symbol_Processed = symbol;
                    if ( symbol->Sym.IfSym.true_line )
                        traverse_line( symbol->Sym.IfSym.true_line->to,output_file )
;
                    if ( symbol->Sym.IfSym.false_line )
                        traverse_line( symbol->Sym.IfSym.false_line->to,output_file)
;
                    return;
        case SET  : make_set( symbol, output_file );
                    break;
        case PAUSE: make_pause( symbol, output_file );
                    break;
        case GOTO : make_goto( symbol, output_file );
                    break;
        case START: make_start( symbol, output_file );
                    break;
        case STOP:  make_stop( symbol, output_file );
                    break;
        case PRINT: make_print( symbol, output_file );
                    break;
        case TEXT : break;
        default   : displayer( "\n\n*** Unknown symbol type\n\n" );
                    break;
    }

    /*
     *  Record the last symbol processed.
     */

    Last_Symbol_Processed = symbol;

    /*
     *  Go the next symbol connected with a line.
     */

    if ( symbol->next )
        symbol = (Symbol *)symbol->next->to;
    else
        symbol = NULL;

    /*
     *  If the next symbol has been generated, the insert a goto into the code
     *  because the code has already been established and that implies that the
     *  code should appear previously in the source file.
     */

        if ( symbol && ( symbol->symbol_generated == GENERATED ) ) {
            fprintf( output_file, "\tgoto L%05d;\n", compute_label_index( symbol ) );
            symbol = NULL;
        }
    }

}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  upd_mode_panel()
*
* MODULE FUNCTION:
*
*   This routine updates uppper left area with user name, etc.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  upd_mode_panel()

{
    Arg         args[1];
    XmString    tcs, tcs2;

    /*
     *  depending on the mode, create the appropriate XmString for the
     *  mode label and the undo button.
     */

    switch ( Mode )
        {
        case AddSymbol  :
                tcs2 = XmStringCreate( "Add Symbol",XmSTRING_DEFAULT_CHARSET );
                tcs = XmStringCreate( "CANCEL", XmSTRING_DEFAULT_CHARSET );
                break;
        case LineDraw   :
                tcs = XmStringCreate( "CANCEL", XmSTRING_DEFAULT_CHARSET );
                tcs2 = XmStringCreate( "Connect Symbols", XmSTRING_DEFAULT_CHARSET );
                break;
        case EditSymbol :
                tcs = XmStringCreate( "UNDO", XmSTRING_DEFAULT_CHARSET );
                tcs2 = XmStringCreate( "Edit Symbol",XmSTRING_DEFAULT_CHARSET );
                break;
        case PrintBox   :
                tcs = XmStringCreate( "CANCEL", XmSTRING_DEFAULT_CHARSET );
                tcs2=XmStringCreate( "Set Print Box",XmSTRING_DEFAULT_CHARSET );
                break;
        case DeleteBox  :
                tcs = XmStringCreate( "CANCEL", XmSTRING_DEFAULT_CHARSET );
                tcs2=XmStringCreate( "Set Delete Box", XmSTRING_DEFAULT_CHARSET );
                break;
        case Help  :
                tcs = XmStringCreate( "CANCEL", XmSTRING_DEFAULT_CHARSET );
                tcs2=XmStringCreate( "Help", XmSTRING_DEFAULT_CHARSET );
                break;
        case MoveBox    :
                tcs = XmStringCreate( "CANCEL", XmSTRING_DEFAULT_CHARSET );
                tcs2=XmStringCreate( "Set Move Box",XmSTRING_DEFAULT_CHARSET );
                break;
        case CopyBox    :
                tcs = XmStringCreate( "CANCEL", XmSTRING_DEFAULT_CHARSET );
                tcs2=XmStringCreate( "Set Copy Box",XmSTRING_DEFAULT_CHARSET );
                break;
        default:
                tcs = XmStringCreate( "CANCEL", XmSTRING_DEFAULT_CHARSET );
                tcs2=XmStringCreate( "Ghost Box",XmSTRING_DEFAULT_CHARSET );
                break;
        }
    XtSetArg( args[0], XmNlabelString, tcs );
    XtSetValues( btn_cancel, args, 1 );
    XtSetArg( args[0], XmNlabelString, tcs2 );
    XtSetValues( txt_mode, args, 1 );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  upd_pos_panel()
*
* MODULE FUNCTION:
*
*   This routine updates the labels in the upper left box.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  upd_pos_panel( purpose_type )

    int purpose_type;

{
    Arg             args[1];
    XmString        tcs;

    /*
     *  Update the status labels.
     */

    tcs = XmStringCreate( Author, XmSTRING_DEFAULT_CHARSET );
    XtSetArg( args[0], XmNlabelString, tcs );
    XtSetValues( txt_author, args, 1 );
    XmStringFree( tcs );

    tcs = XmStringCreate( CompFile, XmSTRING_DEFAULT_CHARSET );
    XtSetArg( args[0], XmNlabelString, tcs );
    XtSetValues( txt_comp, args, 1 );
    XmStringFree ( tcs );

    tcs = XmStringCreate( ElementFile, XmSTRING_DEFAULT_CHARSET );
    XtSetArg( args[0], XmNlabelString, tcs );
    XtSetValues( txt_element, args, 1 );
    XmStringFree ( tcs );

    tcs = XmStringCreate( CreateDate, XmSTRING_DEFAULT_CHARSET );
    XtSetArg( args[0], XmNlabelString, tcs );
    XtSetValues( txt_created, args, 1 );
    XmStringFree ( tcs );

    tcs = XmStringCreate( UpdateDate, XmSTRING_DEFAULT_CHARSET );
    XtSetArg( args[0], XmNlabelString, tcs );
    XtSetValues( txt_last_update, args, 1 );
    XmStringFree ( tcs );

    tcs = XmStringCreate( pPosition, XmSTRING_DEFAULT_CHARSET );
    XtSetArg( args[0], XmNlabelString, tcs );
    XtSetValues( txt_position, args, 1 );
    XmStringFree ( tcs );

    /*
     *  Update the purpose textsw.
     */

    switch ( purpose_type )
        {
        case COMP_PURPOSE :     arm_tgl( tgl_comp );
                                disarm_tgl( tgl_element );
                                break;
        case ELEMENT_PURPOSE :  arm_tgl (tgl_element );
                                disarm_tgl( tgl_comp );
                                break;
        case NO_CHANGE :        break;
        }

    if ( XmToggleButtonGetState(tgl_element) )
        XmTextSetString( txt_purpose, ElementPurpose );
    else
        XmTextSetString( txt_purpose, CompPurpose );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  create_global_variables()
*
* MODULE FUNCTION:
*
*   Function create_global_variables is invoked to create variable declarations for
*   global variables.  It the file pointer passed to the routine is valid, then it is
*   assumed that variables are to be declared (and storage allocated); otherwise they
*   are to be simply EXTERN'ed so that the compiler will know the base data types.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

static int create_global_variables( fp )

    FILE *fp;

{
    char    filename [ MAX_FILENAME ],
            leader   [ MAX_LINE_LENGTH ],
            data_type[ MAX_DATA_TYPE ];
    int     file_close;

    struct symbol_entry *temp;

    /*
     *  If the function was invoked with a valid file pointer (i.e. a Non-NULL value),
     *  it implies that the function is to write the symbol table with no EXTERN
     *  statements (implying that the code is going in the main COMP body to actually
     *  declare storage space).  Otherwise, a .h is created with the CompFile as the
     *  predicate to be included so that successful compilation can occur.
     */

    file_close = NO_CLOSING;
    if ( fp == NULL ) {

        /*
         *  Create the filename to be created.  Open the file for writing.
         */

        sprintf( filename, "%s.h", CompFile );
        if ( ( fp = fopen( filename, "w" ) ) == NULL )
            return( ERR );

        /*
         *  Put a beginning comment in the file.  Establish the leader to be printed
         *  before each variable definition.
         */

        fprintf( fp, "/*\n *  Extern file for COMP: %s\n */\n\n", CompFile );
        strcpy( leader, "extern " );
        file_close = NEEDS_CLOSING;
    }
    else
        /*
         *  Establish that no leader will be printed for the declaration.
         */

        leader[ 0 ] = NULL;

    /*
     *  Scan through the symbol table root (where all globals are defined) and if a
     *  global is found and its use count is greater than zero (which means it has
     *  a current reference), place it in the output file.
     */

    temp = symbol_table;
    while ( temp ) {

        /*
         *  If the entry is a global variable and its accessed, generate
         *  a statement for it.
         */

        if ( ( temp->se_type & VARIABLE ) && ( temp->se_use_count > 0 ) ) {

            /*
             *  Create the string equivalent of the data type and place it in
             *  the output stream.  If the variable is of type CHAR (implying
             *  string then simply declare a character pointer).
             */

            create_C_string_type( ( temp->se_type & SYMBOL_DATA_TYPE ), data_type );

            if ( temp->se_type & CHAR )
                fprintf( fp, "%s%s *%s;\n", leader, data_type, temp->se_symbol );
            else if ( temp->se_type & MATRIX ) {
                    if ( (temp->se_type & WS_GLOBAL) || (temp->se_type & WS_OBJECT) ) {
                        if ( leader[ 0 ] == NULL )
                            fprintf( fp, "%s%s %s[ %d ];\n",
                                         leader,
                                         data_type,
                                         temp->se_symbol,
                                         temp->se_subs[ SUB1 ] );
                        else
                            fprintf( fp,"%s%s *%s;\n",leader,data_type,temp->se_symbol );
                    }
                    else {

                        /*
                         *  Declare an array with the appropriate number of dimensions.
                         */

                        fprintf( fp, "%s%s %s[ %d ]",
                                     leader,
                                     data_type,
                                     temp->se_symbol,
                                     temp->se_subs[ SUB1 ] );
                        switch( temp->se_num_dimensions ) {
                            case 1  : break;
                            case 2  : fprintf( fp, "[ %d ];\n", temp->se_subs[ SUB2 ] );
                                      break;
                            case 3  : fprintf( fp, "[ %d ][ %d ];\n",
                                                   temp->se_subs[ SUB2 ],
                                                   temp->se_subs[ SUB3 ] );
                                      break;
                            case 4  : fprintf( fp, "[ %d ][ %d ][ %d ];\n",
                                                   temp->se_subs[ SUB2 ],
                                                   temp->se_subs[ SUB3 ],
                                                   temp->se_subs[ SUB4 ] );
                                      break;
```

```
                        default: displayer(
                        "Attempted to declare an array with no dimensions specified"
                        );
                    }
                }
            }
            else
                fprintf( fp, "%s%s %s;\n", leader, data_type, temp->se_symbol );
        }  /*  of if  */

        /*
         *  Go to the next symbol in the root of the symbol table.
         */

        temp = temp->se_next;
    }

    /*
     *  Determine if the file needs to be closed.
     */

    if ( file_close == NEEDS_CLOSING )
        fclose( fp );

    /*
     *  Issue a successful return code.
     */

    return( SUCCESS );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  build_comp_file()
*
* MODULE FUNCTION:
*
*   Function build_comp_file is invoked when a COMP file is to be generated, compiled
*   and linked with the appropriate element files.  The function will first generate
*   the COMP main file (which holds global variable declarations and a call to the root
*   element).  The file will then be compiled and linked with all appropriate element
*   files (those elements which exist in the root of the symbol table), the file
*   skeleton_element.o and the math library (to resolve the GCB intrinsic functions).
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

static void build_comp_file( source_fp )

    FILE *source_fp;

{
    char    cc_command[ MAX_OS_COMMAND ],
            skeleton_file[ MAX_PATH ];
    struct symbol_entry *temp;

    /*
     *  Create the header information.
     */

    create_proc_header( source_fp );

    /*
     *  Create the global variables for the main COMP.
     */

    fprintf( source_fp, "/*\n *  Global variables\n */\n\n" );
    create_global_variables( source_fp );

    /*
     *  Create the main routine and place a call to initialize the process table.
     */

    fprintf( source_fp, "\nmain()\n{\n\tinitialize_process_table();\n" );

    /*
     *  Add the statements to bind the any workstation globals or objects to the
     *  appropriate memory address.  The root of the symbol table is scanned for
     *  any of these type of variables.
     */

    temp = symbol_table;
    while ( temp ) {

        /*
         *  If the entry is for a workstation global or for an object (scalar data
         *  only), generate a define statement.
         */
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  upd_status()
*
* MODULE FUNCTION:
*
*   This routine determines if the element is complete and updates the status
*   field accordingly.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  upd_status( complete )

    int complete;

{
    Arg             args[1];
    XmString        tcs;

    if ( complete )
        tcs = XmStringCreate( "Complete",   XmSTRING_DEFAULT_CHARSET );
    else tcs = XmStringCreate( "Incomplete",   XmSTRING_DEFAULT_CHARSET );
    XtSetArg( args[0], XmNlabelString, tcs );
    XtSetValues( txt_status, args, 1 );
    XmStringFree( tcs );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  user_ack()
*
* MODULE FUNCTION:
*
*   This routine popups up a message requiring user acknowledgement and waits until
*   the user answers it.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  user_ack( message )

    char    *message;

{
    Arg         args[1];
    XmString    string;

    /*
     *  Setup up the message to display to the user.
     */

    XtSetArg( args[0], XmNlabelString, XmStringCreate(message,XmSTRING_DEFAULT_CHARSET) )
;
    XtSetValues( lbl_ack, args, 1 );

    /*
     *  Set the size (width) of the user_ack() popup.  This is a pretty ugly
     *  method, but it works.
     */

    elog(3,"setting user_ack width to %i", 200+(8*strlen(message)) );

    XtSetArg( args[0], XmNwidth, (200+(8*strlen(message))) );
    XtSetValues( frm_ack, args, 1 );

    process_popup( dlg_ack, WAIT );
    XtSetArg( args[0], XmNlabelString, &string );
    XtGetValues( lbl_ack, args, 1 );
    XmStringFree( string );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  whirl
*
* MODULE FUNCTION:
*
*   This routine  prints a | or a - in turn to show the progress of the GCB
*   initialization routines.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  whirl()

{
    static int  direction = 0;

    if ( !direction )
        printf("\010\055");
    else
        printf("\010\174");
    fflush( stdout );

    /*
     *  reverse direction of line for next time.
     */

    if ( direction )
        direction = 0;
    else direction = 1;
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  write_defaults()
*
* MODULE FUNCTION:
*
*   This routine  writes the current settings out to the user's defaults file.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.01 - 08/01/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  write_defaults()

{
    FILE    *fp;

    /*
     *  Open the defaults file.
     */

    if (! (fp = fopen(DefaultsFile,"w")) )
        {
        user_ack("Could not open User Defaults file for writing!");
        elog(1,"write_defaults: Could not open User Defaults file for writing!");
        return;
        }

    /*
     *  Go through the options.  Set the values in the defaults file to
     *  to the values of the GCB options.
     *  Opening the file for writing discards the old contents.
     *  Don't mess with the order of options without extreme care.  For example,
     *  the LIB_PATH must preceed the ELEMENT_TYPE and ELEMENT_FILE.  If
     *  changes must be made, check open_read_defaults() for side effects.
     */

    fprintf(fp, "LIBRARY_PATH\t%s\n",  LibPath);

    if (ElementType == ELEMENT)
       fprintf(fp, "ELEMENT_TYPE\tELEMENT\n");
    else
       fprintf(fp, "ELEMENT_TYPE\tLIB\n");

    fprintf(fp, "DISPLAY_FILE\t%s\n",  DisplayFile     );
    fprintf(fp, "COMP_FILE\t%s\n",     CompFile        );
    fprintf(fp, "ELEMENT_FILE\t%s\n",  ElementFile     );
    fprintf(fp, "LOG_FILE\t%s\n",      LogFile         );
    fprintf(fp, "LOG_LEVEL\t%d\n",     ErrorLogLevel   );
    fprintf(fp, "MACROS_PATH\t/GCB/Macros\n");
    fprintf(fp, "OBJECT_TABLE\t%s\n",  MSIDTable       );
    fprintf(fp, "POSITION_PATH\t%s\n", PositionPath    );
    fprintf(fp, "POSITION\t%s\n",      pPosition       );

    if ( TargetLanguage == C )
       fprintf( fp, "TARGET_LANG\tC\n" );
    else if ( TargetLanguage == MOAL )
       fprintf( fp, "TARGET_LANG\tMOAL\n" );
```

```
    if ( temp->se_type & WS_GLOBAL )

        /*
         *  Place a call to the bind routine to bind the variable to a location
         *  in the data acquisition shared memory segment.  If the variable is a
         *  matrix then simply pass the name of the matrix as the binding location.
         */

        if ( temp->se_type & MATRIX )
            fprintf( source_fp, "\tbind_ws_global( \"%s\", %d, %s );\n",
                                        temp->se_symbol,
                                        temp->se_type & SYMBOL_ATTRIBUTES,
                                        temp->se_symbol );
        else
            fprintf( source_fp, "\tbind_ws_global( \"%s\", %d, &%s );\n",
                                        temp->se_symbol,
                                        temp->se_type & SYMBOL_ATTRIBUTES,
                                        temp->se_symbol );

    if ( temp->se_type & WS_OBJECT )

        /*
         *  Place a call to the bind routine to bind the variable to a location
         *  in the data acquisition shared memory segment.  If the variable is a
         *  matrix then simply pass the name of the matrix as the binding location.
         */

        if ( temp->se_type & MATRIX )
            fprintf( source_fp, "\tbind_ws_object( \"%s\", %d, %s );\n",
                                        temp->se_symbol,
                                        temp->se_type & SYMBOL_ATTRIBUTES,
                                        temp->se_symbol );
        else
            fprintf( source_fp, "\tbind_ws_object( \"%s\", %d, &%s );\n",
                                        temp->se_symbol,
                                        temp->se_type & SYMBOL_ATTRIBUTES,
                                        temp->se_symbol );


    /*
     *  Go to the next symbol in the root of the symbol table.
     */

    temp = temp->se_next;
}

/*
 *  Place the call to the root element in the output file and complete the
 *  basic structure of the COMP.
 */

fprintf( source_fp, "\n\t%s();\n}\n", RootElement );

/*
 *  Close the source file and build the cc command to compile and link the COMP
 *  file and its associated ELEMENT files.  All Elements listed in the root of
 *  the symbol table which are used are included on the CC command.
 */

fclose( source_fp );
strcpy( skeleton_file, Swd );
strcat( skeleton_file, SKELETON_OBJECT );
sprintf( cc_command, "cc >/dev/null 2>&1 -o %s %s.c %s",
                    CompFile,
                    CompFile,
                    skeleton_file );

/*
 *  Temporarily increase the use count of the RootElement so that it will
 *  be included in the "cc" line.
 */

if ( increment_symbol_use_count( NULL, RootElement ) == ERR ) {
    displayer( "Could not find root element in the symbol table\n" );
    return;
}  /*  of if  */

temp = symbol_table;
while ( temp ) {

    /*
     *  If the entry is a global variable and it is accessed, generate
     *  a statement for it.
     */

    if ( ( temp->se_type & PROCEDURE ) && ( temp->se_use_count > 0 ) &&
         ( ( temp->se_type & INTRINSIC ) == 0 ) ) {

        /*
         *  If the function accessed is a user defined function, set the path
         *  to where the user defined functions are stored.
         */

        if ( strncmp( temp->se_symbol, "FN_", 3 ) == 0 )

            /*
             *  Append the *.o name with the path to the user defined function
             *  area.
             */

            sprintf( cc_command, "%s %s/%s.o ", cc_command,
                                                UserFuncsPath,
                                                temp->se_symbol );

        else
            {

            /*
             *  We have an Element name, see if its object file is up to date.
             */

            if ( (temp->se_type & INSTALLED) == 0 )
                {
                displayer("\nElement file: %s has not been Installed, build halted",
                            temp->se_symbol );
                displayer("\n\n");

                /*
                 *  We temporarily incremented the user count of the RootElement,
                 *  now decrement.
                 */

                if ( decrement_symbol_use_count(NULL,RootElement) == ERR )
                    {
                    displayer( "Could not find root element in the symbol table\n");
                    elog(1,"build: Could not find root element in the symbol table");
                    }
                return;
                }
```

```
            /*
             *  Append the .o name to the CC line for inclusion in the link process.
             */

            sprintf( cc_command, "%s %s.o ", cc_command, temp->se_symbol );
            }
        }

    /*
     *  Go to the next symbol in the root of the symbol table.
     */

    temp = temp->se_next;
}

/*
 *  Decrease the use count of the RootElement.
 *  in the "cc" line.
 */

if ( decrement_symbol_use_count( NULL, RootElement ) == ERR ) {
    displayer( "Could not find root element in the symbol table\n" );
    return;
}  /*  of if  */

/*
 *  Concatenate on the loading of the math library.
 */

strcat( cc_command, " -lm" );

/*
 *  Compile and link the COMP by invoking the CC compiler through the system
 *  command.  All output from the CC command is dumped to /dev/null.
 */

displayer( "Compiling and link COMP: %s\n", CompFile );

if ( system( cc_command ) )
    {
    displayer( "Compile of COMP: %s failed -- contact GCB support personnel\n",
               CompFile );
    displayer( "Compile command was: %s\n", cc_command );
    }       }
else
    displayer( "Compile of COMP: %s was successful\n", CompFile );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  build_element_file()
*
* MODULE FUNCTION:
*
*   The function build_element_file is invoked to create and compile an element file.
*   The function will create a .c file in the current directory and build the source
*   file by traversing the sysmbol structure created by the GCB user interface.  The
*   C source file created will then be compiled with the -c option so that it may
*   later be linked with a COMP file.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/01/91
*
*****************************************<----->*****************************************/

static void build_element_file( source_fp )

    FILE *source_fp;

{
    int     i;
    char    cc_command[ MAX_OS_COMMAND ],
            skeleton_file[ MAX_PATH ],
            source_line[ MAX_LINE_LENGTH ];
    FILE    *skeleton;

    /*
     *  Invoke the routine to create a include file which contains EXTERNs for all
     *  global variables (this file is then "included" into the source file being
     *  created.
     */

    if ( create_global_variables( NULL ) != 0 ) {
        displayer( "Could not create global reference file -- check permissions\n" );
        return;
    }

    /*
     *  Place an include in the ELEMENT source file to include the globals EXTERN file.
     */

    fprintf( source_fp, "#include \"%s.h\"\n\n", CompFile );

    /*
     *  Mark all entries in the symbol table as not generated.  This bit will prevent
     *  the traverse_list function from being called forever recursively.
     */

    for ( i = 0; i < MAX_SYMBOLS; i++ )
        Symbol_Map[ i ].symbol_generated = NOT_GENERATED;

    /*
     *  Set the posting variable for the end  and the last processed symbol to be NULL.
     */

    End_To_Be_Posted = NULL;
    Last_Symbol_Processed = NULL;
```

```
    /*
     *  Copy the skeleton comp source file to the source file.
     */

    strcpy( skeleton_file, Swd );
    strcat( skeleton_file, SKELETON_ELEMENT );
    if ( (skeleton = fopen(skeleton_file,"r")) == NULL )
        {
        displayer( "\n** Could not open skeleton COMP - notify administrator.\n" );
        displayer( "\n** BUILD HALTED DUE TO UNRECOVERABLE ERROR!\n\n" );
        return;
        }

    while ( fgets( source_line, MAX_LINE_LENGTH, skeleton ) != NULL )
        fprintf( source_fp, "%s", source_line );
    fclose( skeleton );

    /*
     *  Invoke the routine to print procedure header information to the target code
     *  output file.
     */

    create_proc_header( source_fp );

    /*
     *  Traverse the linked list.
     */

    traverse_line( Begin_Sym, source_fp );

    /*
     *  If an end exists to be posted to the output file, output the data.
     */

    if ( End_To_Be_Posted )
        make_trailer( source_fp );

    /*
     *  The source file is now complete.  Close the file and send it to be
     *  compiled.  All output from the compiler is dumped to /dev/null.
     */

    fclose( source_fp );
    sprintf( cc_command, "cc -c %s.c -I%s -I. >/dev/null 2>&1", ElementFile, Swd );
    displayer( "Beginning compilation of: %s\n", ElementFile );

    /*
     *  Display an appropriate error based on the CC command.
     */

    if ( system( cc_command ) )
        {
        displayer( "Compile of Element: %s failed -- contact GCB support personnel\n",
                   ElementFile );
        displayer( "Compile command was: %s\n", cc_command );
        }
    else
        {
        displayer( "Compile of element file %s was successful\n", ElementFile );
        set_sym_attribs( ElementFile, INSTALLED, TRUE );
        update_comp_file( GCompFile, NULL, NULL, NULL, NULL );
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_build()
*
* MODULE FUNCTION:
*
*   Function cbr_build in the entry point into the build process when the GCB user
*   selects that a "install" should occur.  The function will either build an element
*   file or a COMP file depending on the selection of the GCB user.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_build( w, client_data, call_data )

    Widget      w;
    caddr_t     client_data;
    caddr_t     call_data;

{
    char source_file[ MAX_FILENAME ];
    FILE *source_fp;

    /*
     *  Make sure that edit mode is not currently active.
     */

    if (Mode != EditSymbol)
        return;

    /*
     *  If the currently selected target language is UIL, simply return because the
     *  code generation does not support UIL (if the code generation routine was
     *  invoked MANY syntax errors would be reported).  This conditional should be
     *  removed as the GCB matures.
     */

    if ( TargetLanguage == UIL )
        {
        user_ack("Sorry, but UIL is currently not supported");
        return;
        }

    /*
     *  Audit the element.
     */

    show_displayer( True );
    displayer( "Auditing element: %s\n", ElementFile );

    if ( !(audit(LINES_AND_EXPR)) {
        displayer( "\n** Build failed: highlighted symbols are incomplete.  Select\n" );
        displayer( "   Element -> Audit -> Clear Audit to restore symbols.\n" );
        displayer( "\n** BUILD HALTED DUE TO UNRECOVERABLE ERROR!\n\n" );
        return;
     }

    /*
     *  Create the source filename, if the build is for a COMP then then use the COMP
     *  name as the predicate to a .c; otherwise use the ELEMENT name as the predicate.
     */

    if ( (int) client_data == BUILD_COMP )
        strcpy( source_file, CompFile );
    else
        strcpy( source_file, ElementFile );
    strcat( source_file, ".c" );
    displayer( "Building source file: %s\n", source_file );

    /*
     *  Open the target file (for the specified source to be generated).
     */

    if ( ( source_fp = fopen(source_file,"w") ) == NULL ) {
        displayer( "\n** Could not open source file %s - check permissions.\n",
                   source_file );
        displayer( "\n** BUILD HALTED DUE TO UNRECOVERABLE ERROR!\n\n" );
        return;
    }

    /*
     *  Invoke the appropriate routine based on whether or not a COMP or an ELEMENT
     *  build was requested.  The routine invoked will build a source file and compile
     *  the source code.  The two following build routines currently assume only a
     *  C source code file is being generated.
     */

    if ( (int) client_data == BUILD_COMP )
        build_comp_file( source_fp );
    else
        build_element_file( source_fp );
}
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    call_flow.c
*
* FILE FUNCTION:
*
*   Contains the routines which generate and display the Comp Call Flow.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /home/project/3531/Docu/GCB.spec.doc
*
*
* FILE MODULES:
*
*   cbr_call_flow()      - generate and display the Comp Call Flow
*   free_node_list()     - free the linked list built during cbr_call_flow()
*   indent()             - indent a specified number of spaces during call flow output
*   print_node_list()    - display the linked list of element calls
*
*****************************************<----->*****************************************/

#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>

#include "gcb.h"
#include "cbr.h"
#include "widgets.h"
#include "call_flow.h"
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_call_flow()
*
* MODULE FUNCTION:
*
*   This routine generates and displays the Comp Call Flow.  The Comp
*   Call Flow is a list of every element call generated by each element.  The Comp
*   Call Flow starts at the root element and lists every element called by the root,
*   then every element called by the first level elements are listed until the
*   leaves of the element tree are reached.  The comp call flow looks like the
*   following:
*
*       root -> element1 -> elementa
*                           elementb
*               element2 -> elementc
*                           elementd
*
*   Each node of the linked list contains two pointers:
*
*       child -> this will point to the first element called by the current element
*       next  -> this will point to the next element in the list
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_call_flow( w, closure, call_data )

    Widget  w;
    int     closure;
    caddr_t call_data;

{
    char    names[MAX_NODES][MAX_NAME],
            types[MAX_NODES];
    int     cont       = True,
            count,
            dot_count = 0,
            i,
            num_names = 0,
            num_nodes = 0,
            width     = 0;


    /*
     *  Initialize the displayer for CallFlow output.
     */

    show_displayer( True );
    displayer( "\nBuilding comp CallFlow for:\n\n" );
    displayer( "\tComp:         %s\n", CompFile );
    displayer( "\tRoot Element: %s\n", RootElement );
    displayer( "\n\nExtracting calls in Element Files\n" );

    /*
     *  Build a linked list of the element calls.
     */
```

```
    rootPtr = (struct node *) malloc( sizeof(struct node) );
    rootPtr->child = NULL;
    rootPtr->next  = NULL;
    rootPtr->processed = False;
    rootPtr->type      = 'E';
    strcpy( rootPtr->name, RootElement );

    prevPtr = rootPtr;

    /*
     *  Loop through each of the element files called within this comp.  Each
     *  time an element file is opened to extract its list of calls, display
     *  a dot "." to the user so they get some feedback.
     */

    while ( cont )
        {
        displayer( "." );
        dot_count++;
        if ( dot_count == 50 )
            {
            displayer( "\n" );
            dot_count = 0;
            }

        /*
         *  Keep a count of the widest element name so we will be able to
         *  print them out in a nicely formatted fashion.
         */

        prevPtr->processed = True;
        if ( strlen(prevPtr->name) > width )
            width = strlen(prevPtr->name);

        /*
         *  Extract any element calls from the current element file.
         */

        num_names = get_element_calls( names, types, prevPtr->name, prevPtr->type );
        if ( num_names )
            {
            parentPtr = prevPtr;
            i = 0;
            while ( i < num_names )
                {
                if ( num_nodes++ == MAX_NODES )
                    {
                    displayer("\n\n\n*** Number of nodes exceeded!!!\n");
                    displayer("*** Infinite loop possible!\n\n");
                    print_node_list( width );
                    return;
                    }
                currPtr = (struct node *) malloc( sizeof(struct node) );
                currPtr->child     = NULL;
                currPtr->next      = NULL;
                currPtr->processed = False;
                currPtr->type      = types[i];
                strcpy( currPtr->name, names[i] );

                if ( i == 0 )
                    {
                    nextPtr          = currPtr;
                    parentPtr->child = currPtr;
                    }
                else
                    parentPtr->next  = currPtr;
                parentPtr  = currPtr;
                i++;
                }
            currPtr->next = prevPtr;
            prevPtr       = nextPtr;
            parentPtr     = nextPtr;
            }
        else
            {
            while ((prevPtr != rootPtr) && (prevPtr->processed))
                prevPtr = prevPtr->next;
            if ( prevPtr == rootPtr )
                cont = False;
            }
        }

    /*
     *  Print and free the linked list nodes.
     */

    print_node_list( width );
    free_node_list();
}
```

```
/*************************************<----->*************************************
*
* MODULE NAME:  free_node_list()
*
* MODULE FUNCTION:
*
*   This routine frees the linked list of structures built during the generation
*   of the Comp Call Flow.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*************************************<----->*************************************/

void  free_node_list()
{
    currPtr = rootPtr;
    nextPtr = rootPtr;


    /*
     *  If the list is only 1 deep, free the root pointer here.
     */

    if ( rootPtr->child == NULL )
        {
        free( rootPtr );
        return;
        }

    /*
     *  Mark all nodes/structures to a known state.
     */

    do  {
        currPtr->processed = NOT_CLEARED;
        if ((currPtr->child) && (currPtr->child->processed != NOT_CLEARED))
            currPtr = currPtr->child;
        else
            currPtr = currPtr->next;
        }
    while ( currPtr != rootPtr );

    /*
     *  Free all the linked list nodes/structures.
     */

    do  {
        if ((currPtr->processed == CLEARED) || (currPtr->child == NULL))
            {
            nextPtr = currPtr->next;
            free( currPtr );
            currPtr = nextPtr;
            }
        else
            {
            nextPtr = currPtr;
            currPtr = currPtr->child;
            nextPtr->processed = CLEARED;
            }
        }
    while ( currPtr != rootPtr );

    free( rootPtr );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  indent()
*
* MODULE FUNCTION:
*
*   This routine adds a specified number of spaces to the current "displayer" line
*   to make "pretty" output during Display CallFlow.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  indent( num )
{
    int count;

    for ( count=0; count<num; count++ )
        displayer( " " );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  print_node_list()
*
* MODULE FUNCTION:
*
*   This routine prints the Display CallFlow output linked list.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  print_node_list( width )

    int width;

{
    int count = 0;


    width += 4;
    displayer("\n\n\n   CallFlow for Comp: %s\n",CompFile);
    displayer("-------------------------------------------------------------------------\n\n");

    /*
     *  Move through the linked list making sure to indent to the appropriate level.
     */

    currPtr = rootPtr;
    do  {
        displayer("%s",currPtr->name);
        indent( width - strlen(currPtr->name) );
        count += width;
        currPtr->processed = False;
        if ( currPtr->child == NULL )
            {
            /*
             *  If there is an infinite loop, the last child will
             *  not have a child or next pointer.
             */
            if ( currPtr->next == NULL )
                {
                displayer("\n\n");
                return;
                }
            displayer("\n");
            while ((!currPtr->processed) && (currPtr != rootPtr))
                {
                currPtr = currPtr->next;
                if ( count )
                    count -= width;
                }
            indent( count );
            }
        else
            currPtr = currPtr->child;
        }
    while ( currPtr != rootPtr );
```

```
    else
       {
       user_ack("Target Language not set correctly during save of Defaults file");
       elog(1,"Target Language not set correctly during save of Defaults file");
       }

    fprintf(fp, "USER_FUNCS_PATH\t%s\n",    UserFuncsPath );
    fprintf(fp, "WS_GLOBALS\t%s\n",         WSGlobals );
    if ( LogOrCompText )
       fprintf(fp, "DISPLAY_TOGGLE\t1\n" );
    else
       fprintf(fp, "DISPLAY_TOGGLE\t0\n" );

    fclose( fp );
}
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    validate.c
*
* FILE FUNCTION:
*
*   This file contains the routines which perform the Comp Validate function.
*
*
* FILE MODULES:
*
*   cbr_validate()  - perform the Comp Validate function
*
*****************************************<----->*****************************************/

#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>

#include "gcb.h"
#include "widgets.h"
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_validate()
*
* MODULE FUNCTION:
*
*   This routine is called from the main menu to perform the Comp Validate function.
*
*   NOTE: this routine has not yet been implemented.  This routine will be completed
*         when the structure of the Object Access and Work Station Global tables has
*         been defined.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_validate( w, client_data, call_data )

    Widget      w;
    caddr_t     client_data;
    caddr_t     call_data;

{
    show_displayer( True );
    displayer( "   Validating Work Station Globals\n" );
    displayer( "   ------------------------------\n\n" );
    displayer( "Opening Work Station Global table: %s\n\n\n", WSGlobals );

    displayer( "   Validating Object Access\n" );
    displayer( "   ------------------------\n\n" );
    displayer( "Opening Object Access table: %s\n\n\n", MSIDTable );

    displayer( "\n\n***  THIS FEATURE HAS NOT YET BEEN IMPLEMENTED!!  ***\n" );
}
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    var_list.c
*
* FILE FUNCTION:
*
*   This file contains the routines which  pop up if/set symbol creation popups.
*
*
* FILE MODULES:
*
*   all_vars_complete()      - determines if user has provided type for each variable
*   build_var_list()         - builds list of var name/var type pairs from string
*   build_new_var_list()     - ensures that each var in the expr is in symbol table
*   cancel_var_list()        - cancels draw or edit. deletes vars with use count of 0
*   cbr_unknown_type_done()  - adds new variables to the symbol table.
*   cbr_unknown_is_mat( )    - reacts when user (de)selects matrix toggle
*   cbr_unknown_type_select_name() - reacts when user selects variable to specify type
*   cbr_unknown_type_select_type() - reacts when user selects a type for a variable.
*   change_list_use_count()  - ++ or -- use count of each variable in string.
*   check_all_vars()         - deletes all global and local vars with use count <= 0
*   destroy_var_list()       - deallocates the space taken up by a name/type pair list
*   dupe()                   - sees if first name on list is duplicated in list.
*   edit_var_list()          - ++ use counts of vars in new version, -- them in old
*   get_type_from_name()     - converts the string representation of a type to  value
*   is_local()               - uses naming conventions to determine scope of variable
*   keyword()                - determines if word extracted by the scanner is keyword
*   look_on_last_list()      - checks most recently deleted variables for parameter
*   set_last_var_list()      - copies var to list of most recently deleted variables
*   set_type_string()        - sets the type string in the "unknown type" popup
*   string_in_expr()         - determines if parameter expression contains a string.
*
*****************************************<----->*****************************************/

#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include "gcb.h"
#include "symbol.h"
#include "widgets.h"
#include "var_list.h"

#define                     NUM_KEYWORDS 26

int                         last_selected_name = 0,
                            last_selected_type = -1;

struct list_elem            *last_list = NULL;

char                        *type_choices[6] =  {
                            "int",
                            "float",
                            "double",
                            "short",
                            "unsigned",
                            "string"
                            };

char                        *keywords[26] = {
                            "cos",
                            "acos",
                            "sin",
                            "asin",
                            "tan",
                            "atan",
                            "power",
                            "log",
                            "nlog",
                            "exp",
                            "sqrt",
                            "ADD",
                            "SUB",
                            "MULT",
                            "IDENT",
                            "INVERSE",
                            "TRANSP",
                            "CROSS",
                            "DOT",
                            "PI",
                            "not",
                            "and",
                            "or",
                            "bitAnd",
                            "bitOr",
                            "bitXor"
                            };
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  all_vars_complete()
*
* MODULE FUNCTION:
*
*   This routine determines if, in the unknown_vars popup, the user has provided
*   a type for each variable with an unknown type.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  all_vars_complete( num_undecl )

    int num_undecl;

{
    int         i, string_mat, num_conv;
    XmString    tcs;
    Arg         args[1];
    char        *type;
    char        mat[9], temp_of[9], temp_type[9], rows[3], x1[2], x2[9], x3[9], cols[3],

                d3[9], d4[9];

    for ( i=0; i < num_undecl; i++ )
        {

        /*
         *  retrieve label of this variable's type
         */

        XtSetArg( args[0], XmNlabelString, &tcs );
        XtGetValues( lbl_unknown_type_name[i], args, 1 );
        XmStringGetLtoR( tcs, XmSTRING_DEFAULT_CHARSET, &type );

        if ( !strcmp(NULLS, type) )
            {
            user_ack("must assign a type to each variable");
            return( 0 );
            }
        else
            {

            /*
             *  scan the type string
             */

            elog(3,"first word in sscanf: %i\n",    sscanf(type, "%s", mat));
            if ( !strcmp(mat, "matrix") )
                {
                elog(3,"matrix: # of converted items in sscanf: %i\n",
                    num_conv = sscanf(type, "%s %s %s %s %s %s %s %s %s %s",
                    mat, rows, x1, cols, x2, d3, x3, d4, temp_of, temp_type)) ;

                /*
                 *  Do some bounds checking
                 */

                if ( ( atoi(rows) < 1) || (atoi(cols) < 1) ||
                    ( atoi(rows) > 10) || (atoi(cols) > 10) )
                    {
                    user_ack("matrix bounds exceeded");
                    return( 0 );
                    }

                /*
                 *  Don't allow STRING MATRIX
                 */

                string_mat = 0;
                switch( num_conv )
                    {
                    case 6: if (! strcmp(d3, "string") )
                        string_mat = 1;
                        break;
                    case 8: if (! strcmp(d4, "string") )
                        string_mat = 1;
                        break;
                    case 10: if (! strcmp(temp_type, "string") )
                        string_mat = 1;
                        break;
                    }
                if ( string_mat )
                    {
                    user_ack("The GCB currently does not support STRING MATRICES");
                    return( 0 );
                    }
                }
            }
        }
    return( 1 );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  build_var_list()
*
* MODULE FUNCTION:
*
*   This routine builds a list of var name/ var type pairs from the string
*   parameter.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

struct list_elem  *build_var_list( str, elementName )

    char    *str,
            *elementName;

{
    int                 attribs, len, start_index = 0, curr_index, scope;
    struct list_elem    *list_head = NULL, *new_list_elem;
    struct symbol_entry *sym_entry;
    char                temp_name[MAX_NAME];

    /*
     *  If we don't have a string, then we don't have a list of variables.
     */

    if ( !str )
        return( NULL );

    while ( str[start_index] )
        {
        if ( isalpha(str[start_index]) )
            {

            /*
             *  we have found the start of a word
             */

            curr_index = start_index;
            while ( (isalnum(str[curr_index])) || (str[curr_index] == '_') )
                curr_index++;

            /*
             *  extract this word
             */

            len = curr_index - start_index;
            if ( !len ) len = 1;
            strncpy( temp_name, &str[start_index], len );
            temp_name[len] = '\0';

            if ( keyword(temp_name) )
                {

                /*
                 *  we have found a keyword - ignore
                 */

                if ( str[curr_index] )
                    start_index = curr_index + 1;
                else
                    start_index = curr_index;
                }
            else
                {

                /*
                 *  build new list elem structure
                 */

                if (!(new_list_elem=(struct list_elem *)malloc(sizeof(struct list_elem)))
)
                    {
                    user_ack("malloc failed in build var list");
                    exit( ERR );
                    }

                /*
                 *  malloc space for var name
                 */

                if (!(new_list_elem->var_name = (char *)malloc( sizeof (char) * len + 1))
)
                    {
                    user_ack("malloc failed in build var list");
                    exit( ERR );
                    }
                strncpy( new_list_elem->var_name, &str[start_index], len );
                new_list_elem->var_name[len] = '\0';

                /*DEBUG*/
                elog(3,"build var list: %s not keyword, len = %i\n",
                        new_list_elem->var_name, len);

                new_list_elem->var_type = 0;

                /*
                 *  determine scope of new variable
                 */

                attribs = is_local( new_list_elem->var_name );
                if ( attribs & GLOBAL_VAR )
                    scope = 1;
                else scope = 0;

                /*
                 *  if it's in the symbol table, record its type
                 */

                if ( (sym_entry = (struct symbol_entry *)lookup_symbol( scope ?
                    NULL : elementName, new_list_elem->var_name) ) != NULL )
                        new_list_elem->var_type = sym_entry->se_type;
                else
                    elog(3,"build var list: can't find %s %s\n",
                        scope ? "global" : "local", new_list_elem->var_name);

                /*
                 *  add new list element to list
                 */

                new_list_elem->next = list_head;
```

```
            list_head = new_list_elem;

            if ( str[curr_index] )
                start_index = curr_index + 1;
            else
                start_index = curr_index;
            }
        }
        else if ( str[start_index] == '"' )
            {

            /*
             *  skip over everything within quotes
             */

            /*DEBUG*/
            elog(3,"build_var_list: string var");

            start_index++;
            while ( str[start_index] != '"' )
                start_index++;
            start_index++;
            if ( str[start_index] != '\0' )
                elog(3,"build_var_list: string rhs has extra chars at EOL");
            }
        else

            /*
             *  operators, etc. - ignore
             */

            start_index++;
    }

return( list_head );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  build_new_var_list()
*
* MODULE FUNCTION:
*
*   This routine builds a list of var/type pairs and ensures that each var in the
*   expr is in the symbol table with a type. If it isnt, the routine pops up a
*   window that allows the user to input the variable's type.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  build_new_var_list( str, add )

    char    *str;
    int     add;

{
    struct list_elem        *temp_list, *new_list;
    int                     i, undeclared = 0;
    XmString                tcs;
    Arg                     args[1];

    new_list = build_var_list( str, ElementFile );

    /*
     *  see if any of the variables in this expression are undeclared; if they are,
     *  put them in the undeclared popup and post it.
     */

    temp_list = (struct list_elem *)new_list;
    while ( temp_list )
        {

        /*
         *  don't want a var to appear >1 in the list
         */

        if ( (!temp_list->var_type) && (!dupe(temp_list)) )
            {

            /*
             *  undeclared var; set string in next name field to this var's name
             */

            tcs = XmStringLtoRCreate( temp_list->var_name, XmSTRING_DEFAULT_CHARSET );
            XtSetArg( args[0], XmNlabelString, tcs );
            XtSetValues( tgl_unknown_type_name[ undeclared ], args, 1 );
            XmStringFree( tcs );
            undeclared++;
            if ( undeclared > MAX_UNDECLAREDS )
                {
                user_ack("too many undeclareds");
                destroy_var_list( new_list );
                return( 0 );
                }
            }
```

```
        temp_list = (struct list_elem *)temp_list->next;
        }

    destroy_var_list( new_list );
    if ( undeclared )
        {

        /*
         *  unmanage all tgls and labels that are not being used
         *  this time in the popup.
         */

        for ( i=MAX_UNDECLAREDS-1; i >= undeclared; i-- )
            {
            XtUnmanageChild( tgl_unknown_type_name[i] );
            XtUnmanageChild( lbl_unknown_type_name[i] );
            }

        last_selected_name = 0,
        last_selected_type = -1;

        /*
         *  Set defaults for popup: first variable name, type is int,
         *  it isn't a matrix.
         */

        arm_tgl( tgl_unknown_type_name[0] );
        for ( i=1; i < MAX_UNDECLAREDS; i++ )
            disarm_tgl( tgl_unknown_type_name[i] );
        arm_tgl( tgl_unknown_isnt_mat );
        disarm_tgl( tgl_unknown_is_mat );
        XtSetArg( args[0], XmNuserData, undeclared );
        XtSetValues( dlg_unknown_type,args, 1 );

        /*
         *  pop up window to receive types of all undeclareds.
         */

        XtManageChild( dlg_unknown_type );
        return( 0 );
        }
    return( 1 );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_unknown_type_done()
*
* MODULE FUNCTION:
*
*   This routine ensures that every variable in the popup has been given a type,
*   then adds them to the symbol table. It then remanages all the popup components
*   for the next time.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_unknown_type_done( w, closure, call_data )

    Widget                  w;
    int                     closure;
    XmListCallbackStruct    *call_data;

{
    int                     num_conv, j, i, num_undecl, global, attributes = 0,
                            rows, cols, dim3, dim4, num_dim;
    XmString                tcs;
    Arg                     args[1];
    char                    *var_name, *type;
    char                    temp_mat[9], temp_of[9], temp_type[9], r[3], x1[2], x2[9],
                            x3[9], c[3], d3[9], d4[9];

    /*
     *  Give the user a chance to cancel declaration popup.
     */

    if ( !closure )
        {
        reset_unknown_popup();
        return;
        }

    /*
     *  Retrieve the number of undeclared variables from the userData
     *  resource of the unknown type popup.
     */

    XtSetArg( args[0], XmNuserData, &num_undecl );
    XtGetValues( dlg_unknown_type, args, 1 );

    /*
     *  do we have types for all vars in popup?
     */

    if (! all_vars_complete( num_undecl ) )
        return;

    /*
     *  put each variable in the symbol table with its type
     */

    for ( i=0; i < num_undecl; i++ )
```

```
{

/*
 *  get each var name and sscanf its type out of its corresponding
 *  text field.
 */

XtSetArg( args[0], XmNlabelString, &tcs );
XtGetValues( lbl_unknown_type_name[i], args, 1 );
XmStringGetLtoR( tcs, XmSTRING_DEFAULT_CHARSET, &type );
XtSetArg( args[0], XmNlabelString, &tcs );
XtGetValues( tgl_unknown_type_name[i], args, 1 );
XmStringGetLtoR( tcs, XmSTRING_DEFAULT_CHARSET, &var_name );

attributes |= is_local( var_name );

elog(3,"first word in sscanf: %i\n",    sscanf(type, "%s", temp_mat));
if ( !strcmp(temp_mat, "matrix") )
    {

    /*
     *  scan out matrix's dimensions
     */

    elog(3,"# of converted items in sscanf: %i\n",  num_conv = sscanf(type,
        "%s %s %s %s %s %s %s %s %s %s", temp_mat, r, x1, c, x2, d3, x3, d4,
        temp_of, temp_type)) ;
    attributes |= MATRIX;
    rows = atoi( r );
    cols = atoi( c );
    num_dim = 2;
    if ( num_conv > 6 )
        {
        dim3 = atoi( d3 );
        num_dim = 3;
        }
    if ( num_conv > 8 )
        {
        dim4 = atoi( d4 );
        num_dim = 4;
        }
    }
else
    {
    elog(3,"first word in scanf !matrix: %i\n", sscanf(type,"%s",temp_type));
    num_dim = rows = cols = dim3 = dim4 = 0;
    }
if ( num_dim == 2 )
    attributes |= get_type_from_name( d3 );
else if ( num_dim == 3 )
    attributes |= get_type_from_name( d4 );
else
    attributes |= get_type_from_name( temp_type );

if ( attributes & MATRIX )
    elog(3,"unknown done: MATRIX!  rows %i, cols %i\n", rows, cols);

elog(3,"adding %s symbol %s with attribs %i and r %i c %i d3 %i d4 %i\n",
    (attributes & GLOBAL_VAR) ? "global ":"local ", var_name, attributes,
    rows, cols, dim3, dim4 );

/*
 *  add new var to symbol table with proper attribs and scope
 */

    if ( (j = add_symbol_entry( (attributes & GLOBAL_VAR) ? NULL : ElementFile,
        var_name, attributes,num_dim,  rows, cols, dim3, dim4)) != SUCCESS )
            {
            elog(3,"add unknown symbol entry failed, returned %i", j);
            user_ack("add unknown symbol entry failed");
            exit( ERR );
            }
    attributes = rows = cols = 0;
    }

/*
 *  remanage and reset all tgls and lbls for the next time
 */

reset_unknown_popup();
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_unknown_type_select_name()
*
* MODULE FUNCTION:
*
*   This routine reacts when the user selects a variable to specify its type.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_unknown_type_select_name( w, closure, call_data )

    Widget                          w;
    int                             closure;
    XmToggleButtonCallbackStruct    *call_data;

{
    if ( call_data->reason == XmCR_DISARM )
        return;

    /*
     *  record last selected type to update text string of var being defined.
     */

    last_selected_name = closure;
    elog(3,"last selected is %d\n", last_selected_name = closure);
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME   cbr_unknown_is_mat()
*
* MODULE FUNCTION:
*
*   This routine reacts when the user selects or deselects the matrix toggle
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_unknown_is_mat( w, closure, call_data )

    Widget                          w;
    int                             closure;
    XmToggleButtonCallbackStruct    *call_data;

{
    /*
     *  accept only arm toggle calls
     */

    if ( call_data->reason == XmCR_DISARM )
        return;

    /*
     *  update type string of var being defined.
     */

    set_type_string( closure );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_unknown_type_select_type()
*
* MODULE FUNCTION:
*
*   This routine reacts when the user selects a type for a variable.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_unknown_type_select_type( w, closure, call_data )

    Widget                  w;
    int                     closure;
    XmListCallbackStruct    *call_data;

{
    if ( call_data->reason == XmCR_DISARM )
        return;
    last_selected_type = closure;

    /*
     *  reset type string depending on whether or not matrix is set.
     */

    if ( XmToggleButtonGetState(tgl_unknown_is_mat) )
        set_type_string( 0 );
    else set_type_string( 1 );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cancel_var_list()
*
* MODULE FUNCTION:
*
*   This routine cancels a draw or edit in progress.  It checks all global and
*   local variables: if any have a use count of 0, it decrements their use count.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  cancel_var_list()

{
    /*DEBUG*/
    elog(3,"cancel var list: checking all vars");

    /*
     *  must remove from the symbol table all vars introduced during this
     *  expression's creation, so they may be reused.
     */

    check_all_vars();
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  change_list_use_count()
*
* MODULE FUNCTION:
*
*   This routine creates a list of name/type pairs from the parameter string;
*   it then increments or decrements the use count of each variable. If a
*   variable whose use count is to be incremented is not in the symbol table,
*   its symbol must have been deleted and is now being restored; look on the list
*   of variables from the last symbol to be deleted.
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  change_list_use_count( sym, str, incr_decr )

    Widget  sym;
    char    *str;
    int     incr_decr;

{
    struct list_elem        *temp_list;
    struct  symbol_entry    *entry;
    int                     i, local, scope;

    if (! (temp_list = build_var_list(str,ElementFile)) )
        {
        /*DEBUG*/
        elog(3,"change u.c.: build var list returned NULL");
        return;
        }

    while ( temp_list )
        {
        /*DEBUG*/
        elog(3,"changing use count for %s", temp_list->var_name);

        /*
         *  determine scope of variable
         */

        local = is_local(temp_list->var_name);
        if ( local & GLOBAL_VAR )
            scope = 1;
        else scope = 0;

        if ( entry = (struct symbol_entry *)lookup_symbol(scope ? NULL :
            ElementFile, temp_list->var_name) )
                {
                if ( incr_decr )
                    i = increment_symbol_use_count( scope ? NULL : ElementFile,
                        temp_list->var_name );
                else i = decrement_symbol_use_count( scope ? NULL : ElementFile,
                    temp_list->var_name );
                }
        else
            {
elog(3,"can't find %s %s\n",  scope ? "global" : "local", temp_list->var_name);
            look_on_last_list( temp_list->var_name, local );
            }
        temp_list = (struct list_elem *)  temp_list->next;
        }
    destroy_var_list( temp_list );

    /*
     *  user may have introduced extraneous variables during this expression
     *  creation - delete them from the symbol table.
     */

    check_all_vars();
}
```

```
/*****************************************<-----*****************************************
*
* MODULE NAME:  check_all_vars()
*
* MODULE FUNCTION:
*
*   This routine goes through all global and local vars; deletes those with
*   a use count of 0 or less.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<-----*****************************************/

int  check_all_vars()

{
    struct symbol_entry     *varlist;

    varlist = (struct symbol_entry *) lookup_local_varlist( ElementFile );

    while ( varlist )
        {
elog(3,"check all vars: checking use count of local var %s\n",  varlist->se_symbol);

        /*
         *  Skip over procedures and intrinsics
         */

        if ( (varlist->se_type & VARIABLE) && (varlist->se_use_count <= 0) )
            {
elog(3,"local var %s has u.c. <= 0, decrementing\n", varlist->se_symbol);
            decrement_symbol_use_count(ElementFile, varlist->se_symbol);
            }
        varlist =  (struct symbol_entry *)varlist->se_next;
        }

    varlist = (struct symbol_entry *) lookup_global_varlist();

    while ( varlist )
        {
elog(3,"check all vars: checking use count of global var %s\n",  varlist->se_symbol);
        if ( (varlist->se_type & VARIABLE) && (varlist->se_use_count <= 0) )
            {
elog(3,"global var %s has u.c. <= 0, decrementing\n", varlist->se_symbol);
            decrement_symbol_use_count( NULL, varlist->se_symbol );
            }
        varlist =  (struct symbol_entry *)varlist->se_next;
        }

    destroy_global_varlist( varlist );
}
```

```
/*****************************************<-----*****************************************
*
* MODULE NAME:  destroy_var_list()
*
* MODULE FUNCTION:
*
*   This routine deallocates the space taken up by a name/type pair list.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<-----*****************************************/

void  destroy_var_list( list )

    struct list_elem    *list;

{
    struct list_elem    *temp = list;

    while ( temp )
        {
        temp = temp->next;
        free( list->var_name );
        free( list );
        list = temp;
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  dupe()
*
* MODULE FUNCTION:
*
*   This routine returns true if the first name on the parameter list is duplicated
*   elsewhere in the list.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  dupe( list )

    struct list_elem *list;

{
    struct list_elem    *temp_list;
    char                name[20];

    strcpy( name, list->var_name );
    temp_list = (struct list_elem *) list->next;

    while ( temp_list )
        {
        if ( !strcmp(temp_list->var_name, name) )
            return( 1 );
        temp_list = (struct list_elem *) temp_list->next;
        }
    return( 0 );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  edit_var_list()
*
* MODULE FUNCTION:
*
*   This routine is called when a symbol has been edited; it increments the use
*   counts of the variables in the new version, decrements them in the old.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  edit_var_list( old_str, new_str )

    char    *old_str, *new_str;

{
    elog(3," edit var list: incrementing use count for string %s \n", new_str);
    change_list_use_count( NULL, new_str, 1 );
    elog(3," edit var list: decrementing use count for string %s \n", old_str);
    change_list_use_count( NULL, old_str, 0 );
    check_all_vars();
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  get_type_from_name()
*
* MODULE FUNCTION:
*
*   This routine converts the string representation of a type to a value.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  get_type_from_name( temp_type )

    char    *temp_type;

{
    if ( !strcmp(temp_type, "int") )
         return( INTEGER );
    else if ( !strcmp(temp_type, "float") )
         return( FLOAT );
    else if ( !strcmp(temp_type, "unsigned") )
         return( UNSIGNED );
    else if ( !strcmp(temp_type, "double") )
         return( DOUBLE );
    else if ( !strcmp(temp_type, "short") )
         return( SHORT );
    else if ( !strcmp(temp_type, "string") )
         return( CHAR );
    else
        {
        user_ack("get_type from name: unknown type");
        return( ERR );
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  is_local()
*
* MODULE FUNCTION:
*
*   This routine uses the naming conventions to determine the scope of a variable.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  is_local( str )

    char    *str;

{
    int     attributes = 0;

    /*
     *  User_defined functions start with FN_
     */

    if ( (str[0] == 'F')  &&
         (str[1] == 'N')  &&
         (str[2] == '_')  )
            attributes |= PROCEDURE;
    else attributes |= VARIABLE;

    /*
     *  Objects start with V
     */

    if ( str[0] == 'V' )
        attributes |= WS_OBJECT;

    /*
     *  Workstation parameters start with WS_
     */

    else if ( (str[0] == 'W')  &&
              (str[1] == 'S')  &&
              (str[2] == '_')  )
            attributes |= WS_GLOBAL;

    /*
     *  Globals start with GV_
     */

    else if ( (str[0] == 'G')  &&
              (str[1] == 'V')  &&
              (str[2] == '_')  )
            attributes |= GLOBAL_VAR;

    /*
     *  everything except user-def funs that is not an object, ws param,
     *  or global is a local.
     */
```

```
    else if ( attributes & VARIABLE )
        attributes |= LOCAL_VAR;

    /*
     *  everything that is not a local is also a global.
     */

    if ( !(attributes & LOCAL_VAR) )
        attributes |= GLOBAL_VAR;

    return( attributes );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  keyword()
*
* MODULE FUNCTION:
*
*   This routine determines if the word extracted by the scanner is a keyword.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  keyword( name )

    char    *name;

{
    int     i;

    /*
     *  if we find the parameter in the keywords array, return true, else false.
     */

    for ( i=0; i<NUM_KEYWORDS; i++ )
        if ( !strcmp(keywords[i], name) )
            return( 1 );

    return( 0 );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  look_on_last_list()
*
* MODULE FUNCTION:
*
*   This routine checks the list of most recently deleted variables for the
*   parameter name.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  look_on_last_list( str, local )

    char    *str;
    int     local;

{
    struct list_elem    *temp = last_list;
    int                 scope;

    while ( temp )
        {

        /*
         *  go through list of most recently deleted vars; find the parameter
         *  and increment its use count
         */

        if ( !strcmp(temp->var_name, str) )
            {
            if ( local & LOCAL_VAR )
                scope = 0;
            else scope = 1;

            /*
             *  add var back into symbol table.
             */

            if ( (add_symbol_entry( scope ? NULL : ElementFile, temp->var_name,
                temp->var_type, temp->rows, temp->cols)) != SUCCESS )
                    {
                    user_ack("look on last list: add local symbol entry failed");
                    exit( ERR );
                    }
            elog(3,"look on last list: added sym %s with type %i\n",
                temp->var_name, temp->var_type);
            increment_symbol_use_count( scope ? NULL : ElementFile, temp->var_name);
            return;
            }
        else
            temp = (struct list_elem *) temp->next;
        }
    elog(3,"never found symbol %s on last list", str);
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  reset_unknown_popup()
*
* MODULE FUNCTION:
*
*   This routine
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  reset_unknown_popup()

{
    int         i;
    XmString    tcs = XmStringLtoRCreate( NULLS,   XmSTRING_DEFAULT_CHARSET );
    Arg         args[1];

    for ( i=0; i < MAX_UNDECLAREDS; i++ )
        {
        XtManageChild( tgl_unknown_type_name[i] );
        XtManageChild( lbl_unknown_type_name[i] );
        XtSetArg( args[0], XmNlabelString, tcs );
        XtSetValues( lbl_unknown_type_name[i], args, 1 );
        }
    XmStringFree( tcs );

    XtUnmanageChild( dlg_unknown_type );
    XmTextSetString( txt_mat_numrows, NULLS );
    XmTextSetString( txt_mat_numcols, NULLS );
    XmTextSetString( txt_mat_dim3, NULLS );
    XmTextSetString( txt_mat_dim4, NULLS );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  set_last_var_list()
*
* MODULE FUNCTION:
*
*   This routine sets the list of most recently deleted variables to be the
*   parameter list.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  set_last_var_list( str )

    char    *str;

{
    /*
     *  retain list of most recently deleted vars in case of undo.
     */

    if ( last_list )
        destroy_var_list( last_list );
    last_list = build_var_list( str, ElementFile );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  set_type_string()
*
* MODULE FUNCTION:
*
*   This routine sets the type string in the "unknown type" popup
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  set_type_string( matrix )

    int matrix;

{
    XmString        tcs;
    Arg             args[2];
    char            temp[50];

    if ( last_selected_type < 0 )

        /*
         *  user has not yet selected a type.
         */

        return( 0 );
    if ( !matrix )
        {

        /*
         *  if they're not there, assign defaults to numrows and cols
         */

        strcpy( temp, XmTextGetString(txt_mat_numrows));
        if ( ! temp[0] )
             XmTextSetString( txt_mat_numrows, "2" );
        strcpy( temp, XmTextGetString(txt_mat_numcols) );
        if (! temp[0] )
             XmTextSetString( txt_mat_numcols, "2" );
        strcpy( temp, "matrix " );
        strcat( temp, XmTextGetString(txt_mat_numrows) );
        strcat( temp, " x " );
        strcat( temp, XmTextGetString(txt_mat_numcols) );

        /*
         *  if they've been specified, add 3rd and 4th dims to string.
         */

        if ( atoi(XmTextGetString(txt_mat_dim3)) )
            {
            strcat( temp, " x " );
            strcat( temp, XmTextGetString(txt_mat_dim3) );
            }

        if ( atoi(XmTextGetString(txt_mat_dim4)) )
            {
            strcat( temp, " x " );
```

```
            strcat( temp, XmTextGetString(txt_mat_dim4) );
            }

        strcat( temp, " of " );
        strcat( temp,  type_choices[last_selected_type] );
        }
    else strcpy( temp,  type_choices[last_selected_type] );

    /*
     *  replace string next to variable name with new type.
     */

    tcs = XmStringLtoRCreate( temp,   XmSTRING_DEFAULT_CHARSET );
    XtSetArg( args[0], XmNlabelString, tcs );
    XtSetValues( lbl_unknown_type_name[ last_selected_name ], args, 1);
    XmStringFree( tcs );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  string_in_expr()
*
* MODULE FUNCTION:
*
*   This routine determines if the parameter expression contains a string.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  string_in_expr( string )

    char    *string;

{
    int     i, j;

    if ( string == NULL )
        return( 0 );

    j = strlen( string );
    for ( i=0; i<j; i++ )
        {
        if ( string[i] == '"' )
            return( 1 );
        }
    return( 0 );
}
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    var_list.h
*
* FILE FUNCTION:
*
*   This file contains the global variables and function prototypes for var_list.c
*
*
* SPECIFICATION DOCUMENTS:
*
*   /home/project/3531/Docu/GCB.spec.doc
*
*
* FILE MODULES:
*
*   N/A
*
*****************************************<----->*****************************************/

/*
 *  This structure is used during the building of a list of variables which are
 *  contained in an expression.
 */

struct list_elem
    {
    int                 var_type;
    char                *var_name;
    int                 rows,
                        cols;
    struct list_elem    *next;
    };

/*
 *  Function prototypes for var_list.c
 */

struct list_elem    *build_var_list();
void                destroy_var_list();
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    widgets.h
*
* FILE FUNCTION:
*
*   This file contains the all the widget declarations for the Comp Builder.  Most
*   widget names use the following convention:
*
*       btn_    - push button/command button widget
*       dlg_    - dialog    widget, usually popup dialog
*       frm_    - form      widget
*       lbl_    - lable     widget
*       pix_    - pixmap    widget
*       sep_    - separator widget
*       txt_    - text edit widget
*
*   This file also contains the function prototypes for the widget creation "helper"
*   routines.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /home/project/3531/Docu/GCB.spec.doc
*
*
* FILE MODULES:
*
*   N/A
*
*
*****************************************<----->*****************************************
/

/*
 *  GCB high level widgets - main form, main window, and others.
 */

Display         *display;

XtAppContext    app_context;

Widget          frm_container,
                mnb_main,
                top,
                win_main;

/*
 *  Popup widgets and main screen widgets..
 */

Widget  CancelW, DoneW, FormW, HelpW, BtnW, ResetW,

        bboard, btn_cancel,
        btn_color_template, btn_ask_y, btn_ask_n, btn_ask_h,
        btn_cre_pos_show, btn_logic_attribs, btn_logic_parse,

        cascade,
        draw_area,

        dlg_help, dlg_logic, dlg_call, dlg_pause, dlg_copy_elem,
        dlg_text, dlg_color, dlg_ask, dlg_ack, dlg_purpose, dlg_sel_comp,
        dlg_sel_elem, dlg_cre_elem, dlg_cre_comp, dlg_sel_comp, dlg_print_elem,
        dlg_del_elem, dlg_cre_pos, dlg_sel_pos, dlg_tl_select, dlg_var_input, dlg_file,
        dlg_help, dlg_dis_status, dlg_logic_attribs, dlg_def_fn, dlg_logic_type,
        dlg_var_type, dlg_other_input, dlg_displayer, dlg_sel_root,
        dlg_unknown_type, dlg_str_input, dlg_start,

        eq_btn,

        frame_call, frame_math_menu, frame_sym, frame_palette,  frame_mbar,
        frame_status, frame_cancel, frame_WA, frame_matrix_menu, frame_trig_menu,

        frm_math_menu, frm_status, frm_WA, frm_cancel, frm_sym,
        frm_logic, frm_ask, frm_main, frm_ack, frm_mbar, frm_matrix_menu,
        frm_def_fn, frm_other_input, frm_var_type, frm_trig_menu,

        lbl_call_header, lbl_ack, lbl_text_header, lbl_color_header, lbl_ask,
        lbl_val_func_path, lbl_cre_pos_ne, lbl_cre_pos_ppath, lbl_start_hdr,
        lbl_var_input_header, lbl_tl_header, lbl_sel_pos_path,
        lbl_file, lbl_val_pname, lbl_val_cname, lbl_val_ename, lbl_val_cwd,
        lbl_val_err_file, lbl_val_lib_path, lbl_val_mac_path, lbl_val_msid,
        lbl_val_ws_glob, lbl_val_snap, lbl_val_audit, lbl_val_language, lbl_val_user,
        lbl_val_type, lbl_val_time, lbl_val_date, lbl_val_cflow,

        list_elem, list_del_elem, list_sel_elem, list_sel_comp, list_sel_pos,
        list_sel_pos, list_sel_root, list_comp,

        Lmenu_popup, Rmenu_popup, menu_pane, menu_pullrite, menu_pullrite2, menu_item,

        pix_help,

        rb_del_elem, rb_cre_elem, rb_tl_select, rb_purpose,
        rc_color, rb_sel_elem, rb_print_elem, rb_copy_elem, rc_pause_num, rc_pos,
        rb_call, rb_color, rb_sym_size, rb_font_size, rb_var_type, rb_isit_mat,
        rb_unknown_type_name, rb_unknown_type_type, rb_unknown_isit_mat,

        rc_pause_units, rc_help, rc_logic, rc_var, rc_math, rc_oper, rc_math_menu,
        rc_palette, rc_rel, rc_color_iname, rc_color_template, rc_def_fn, rc_expr_num,
        rc_mat, rc_trig,

        scr_cre_comp, scr_WA, scr_comment, scr_help, scr_logic, scr_expr, scr_text,
        scr_cre_elem, scr_purpose, scr_var_input, scr_displayer,

        sampleSymbol, sel_bx_def_fn, sel_bx_var_type,

        tgl_comp, tgl_element, tgl_call_ce, tgl_call_le,
        tgl_color_back, tgl_color_fore, tgl_sel_elem_le, tgl_sel_elem_ce,
        tgl_cre_elem_le, tgl_del_elem_le, tgl_del_elem_ce, tgl_cre_elem_ce,
        tgl_print_elem_red, tgl_print_elem_norm, tgl_language_c, tgl_language_moal,
        tgl_language_uil, tgl_copy_elem_ce, tgl_copy_elem_le, tgl_sym_size_sm,
        tgl_sym_size_lg, tgl_font_size_lg, tgl_font_size_sm, tgl_var_type_int,
        tgl_var_type_double, tgl_var_type_real,  tgl_var_type_string,
        tgl_var_type_unsigned, tgl_var_type_short, tgl_is_mat, tgl_isnt_mat,
        tgl_unknown_type_int, tgl_unknown_type_float,  tgl_unknown_type_double,
        tgl_unknown_type_short, tgl_unknown_type_unsigned,  tgl_unknown_type_string,
        tgl_unknown_is_mat,  tgl_unknown_isnt_mat,

        txt_author, txt_comp, txt_created, txt_element, txt_call_ne, txt_start,
        txt_print_elem_nc, txt_pause_value, txt_last_update, txt_mode, txt_position,
        txt_purpose, txt_status, txt_cre_elem_ne, txt_cre_comp_name, txt_del_el,
        txt_cre_comp_re_name, txt_cre_pos_ppath, txt_cre_pos_ne, txt_sel_pos,
        txt_cre_pos_nd, txt_copy_elem_ne, txt_file, h_scroll, txt_sel_root,
        txt_matdim_x, txt_matdim_y, txt_matdim_3, txt_matdim_4, txt_matelm_3,
        txt_matelm_4, txt_str_input, txt_matelm_x, txt_matelm_y, txt_var_name,
        txt_mat_numrows, txt_mat_numcols, txt_mat_dim3, txt_mat_dim4, txt_def_fn,
        txt_print_elem_pname,
```

```
        v_scroll, h_rule, v_rule;

/*
 *  Various widget arrays.
 */

Widget  help[6],
        lbl_unknown_type_name[5],
        menu_list[7],
        symbols[MAX_SYMBOLS],
        tgl_unknown_type_name[5],
        w[41];

Widget  symbol;

/*
 *  Function prototypes for widget creation "helper" routines.
 */

Widget  Ncr_command(),      /* Ncr_* routines are temporary replacements for cr_*   */
        Ncr_rel_cmd(),      /* routines.  The Ncr_* versions accept XtCallbackProcs */
        Ncr_toggle(),       /* instead of XtCallback lists.                         */
        cr_cascade(),
        cr_command(),
        cr_form(),
        cr_frame(),
        cr_label(),
        cr_list(),
        cr_pixmap(),
        cr_popup(),
        cr_pulldown(),
        cr_radio_box(),
        cr_rel_cmd(),
        cr_rowcol(),
        cr_scr_text(),
        cr_separator(),
        cr_text(),
        cr_toggle();
```

```
# define L_PERIN 257
# define R_PERIN 258
# define REL_OPER 259
# define ID 260
# define WSID 261
# define OBID 262
# define MUL 263
# define DIV 264
# define SIGN 265
# define FIXED_FUNC 266
# define USER_FUNC 267
# define STRING 268
# define POWER 269
# define PERIOD 270
# define COMMA 271
# define LOG_OPER 272
# define NOT 273
# define UNSIGNED_REAL 274
# define UNSIGNED_HEX 275
# define UNSIGNED_INT 276
# define SET_EQ 277
# define EQEQ 278
# define MATRIX_OPER 279
# define L_BRACK 280
# define R_BRACK 281
# define MADD 282
# define MSUB 283
# define MMULT 284
# define IDENT 285
# define INVERS 286
# define TRANSP 287
# define CROSS 288
# define DOT 289
# define PI 290
# define BIT_OPER 291
# define SHIFT 292
# define ADD 293
# define MINUS 294
# define UNARY 295
```

```
%term   L_PERIN R_PERIN REL_OPER ID WSID OBID MUL DIV SIGN FIXED_FUNC USER_FUNC STRING P
OWER
%term   PERIOD COMMA LOG_OPER NOT UNSIGNED_REAL UNSIGNED_HEX UNSIGNED_INT SET_EQ EQEQ
%term   MATRIX_OPER L_BRACK R_BRACK MADD MSUB MMULT IDENT INVERS TRANSP CROSS DOT PI
%term   BIT_OPER SHIFT
%left   ADD MINUS
%left   MUL DIV
%left   BIT_OPER SHIFT
%right  UNARY
%%
expr            :   /* expression must match symbol type */

                    if_expr             {
                                          if ( SetSym )
                                            {
                                            parse_error_value = SYNTAX_ERR;
                                            YYERROR;
                                            }
                                        }
                |   set_expr            { if ( !SetSym )
                                            {
                                            parse_error_value = SYNTAX_ERR;
                                            YYERROR;
                                            }
                                        }
                ;

set_expr        :   identifier          { set_state(7, 1); }
                    SET_EQ              { WhereAmI = RHS; set_state(5, 1); }
                    set_expr_rhs        { stable_state = 1; }
                |   identifier          { }
                    matrixaccess        { set_state(7, 1);}
                    SET_EQ              { WhereAmI = RHS; set_state(5, 1); }
                    set_expr_expr       { stable_state = 1; }
                ;

set_expr_expr   :   simple_expr         { set_state(2,1); }
                ;

set_expr_rhs    :   simple_expr         { set_state(16,1); }
                |   matrix              {  set_state(16,1); }
                |   shift_expr
                ;

if_expr         :   simple_expr         { set_state(10, 0); }
                    REL_OPER            { WhereAmI=RHS; set_state(5,1); }
                    simple_expr         { set_state(2, 0); stable_state = 1; }
                |   simple_expr         { set_state(10, 0); }
                    EQEQ                { WhereAmI=RHS; set_state(5,1); }
                    simple_expr         { set_state(2, 0);  stable_state = 1; }
                |   if_expr             { set_state(9, 0); }
                    LOG_OPER            { WhereAmI=LHS; set_state(5,1); }
                    if_expr             { set_state(9, 0); stable_state = 1; }
                ;

simple_expr     :   terminator          { set_state(2, 0); }
                |   simple_expr         { set_state(11, 0); }
                    operator            { }
                    terminator          { set_state(2, 0); }
                ;

shift_expr      :   simple_expr         { set_state(11, 0); }
                    SHIFT               { set_state(23, 1); save_operator( SHIFT_OPER );
                                          shift_value       { init_inputs( -3 ); }
                ;

operator        :   add_op              { }
                |   mul_op              { }
                ;

add_op          :   ADD                 { set_state(5, 1); save_operator( ADD_OPER ); }
                |   MINUS               { set_state(5, 1); save_operator( MINUS_OPER ); }
                |   BIT_OPER            { set_state(5, 1); }
                ;

mul_op          :   MUL                 { set_state(5, 1); save_operator( MULT_OPER ); }
                |   DIV                 { set_state(5, 1); save_operator( DIV_OPER ); }
                ;

matrix          :   factor              { }
                |   IDENT               { save_operator( IDENT_OPER ); }
                |   INVERS              { }
                    identifier          { save_operator(INVER_OPER); }
                |   TRANSP identifier   { save_operator( TRANS_OPER ); }
                |   identifier          { }
                    matrixoper          { }
                    identifier          { }
                ;

matrixaccess    :   vectoraccess        { process_matrix_dimensions(); }
                |   double              { process_matrix_dimensions(); }
                |   triple              { process_matrix_dimensions(); }
                |   triple              { }
                    vectoraccess        { process_matrix_dimensions(); }
                ;

vectoraccess    :   L_BRACK             { set_bracket(); }
                    UNSIGNED_INT
                    R_BRACK             { }
                ;

double          :   vectoraccess        { }
                    vectoraccess        { }
                ;

triple          :   double              { }
                    vectoraccess        { }
                ;

matrixoper      :   MADD                { save_operator( MADD_OPER ); }
                |   MSUB                { save_operator( MMINUS_OPER ); }
                |   MMULT               { save_operator( MMULT_OPER ); }
                |   CROSS               { save_operator( CROSS_OPER ); }
                |   DOT                 { save_operator( DOT_OPER ); }
                ;

terminator      :   factor              { set_state(2, 0); }
                |   factor
                    matrixaccess        { set_state(2, 0); }
                ;

factor          :   NOT                 { set_state(5, 1); }
                    factor              { set_state(2, 0); }
                |   function_id         { set_state(2, 0); }
                |   L_PERIN             { paren_count++; set_state(5, 1); }
                    simple_expr         { set_state(1, 0); }
                    R_PERIN             { paren_count--; set_state(2, 1); }
```

PRECEDING PAGE BLANK NOT FILMED

```
                |   UNSIGNED_INT        { set_state(2, 1); save_type( INTEGER ); }
                |   sign                { set_state(12, 1); }
                    UNSIGNED_INT        { set_state(2, 1); save_type( INTEGER ); }
                |   UNSIGNED_REAL       { set_state(2, 1); save_type( FLOAT ); }

                |   sign                { set_state(12,1);}
                    UNSIGNED_REAL       { set_state(2, 1); save_type( FLOAT ); }
                |   UNSIGNED_HEX        { set_state(2, 1); save_type( INTEGER ); }

                |   sign                { set_state(12, 1); }
                    UNSIGNED_HEX        { set_state(2, 1); save_type( INTEGER ); }
                |   string              { set_state(2, 1); save_type ( CHAR ); }
                |   identifier          { set_state(20, 1); }
                |   PI                  { set_state(20, 1); }
                ;

function_id     :   FIXED_FUNC          { set_state(13, 1);
                                          set_func_type( GCB_DEFINED ); }
                    L_PERIN             { paren_count++; set_state(19, 1); func_state( O
N ); }
                    param               { set_state(18, 1); }
                    R_PERIN             { paren_count--; set_state(2, 1);  func_state( O
FF ); }
                |   POWER               { set_state(13, 1);
                                          set_func_type( GCB_DEFINED ); }
                    L_PERIN             { paren_count++; set_state(19, 1); func_state( O
N ); }
                    params              { set_state(14, 1); }
                    R_PERIN             { paren_count--; set_state(2, 1); func_state( OF
F ); }
                |   USER_FUNC           { set_state(13, 1);
                                          set_func_type( USER_DEFINED ); }
                    L_PERIN             { paren_count++; set_state(15, 1); func_state( O
N ); }
                    param_list
                    R_PERIN             { paren_count--; set_state(2, 1); func_state( OF
F ); }
                ;

shift_value     :   param               { }
                |   USER_FUNC           { set_state(13, 1);
                                          set_func_type( USER_DEFINED ); }
                    L_PERIN             { paren_count++; set_state(15, 1); func_state( O
N ); }
                    param_list
                    R_PERIN             { paren_count--; set_state(2, 1); func_state( OF
F ); }
                ;

param_list      :                       { set_state(21, 1); }
                |   params              { set_state(14, 0); }
                ;

params          :   params              { set_state(14, 0); }
                    COMMA               { set_state(15, 1); }
                    params              { set_state(14, 0); }
                |   param               { set_state(14, 1); }
                ;

param           :   identifier          { }
                |   UNSIGNED_INT        { }
                |   UNSIGNED_REAL       { }
                |   UNSIGNED_HEX        { }
                ;

identifier      :   ID                  { }
                |   WSID                { }
                |   OBID                { }
                ;

string          :   STRING              { }
                ;

sign            :   MINUS %prec UNARY   { }
                ;


%%
/*
 *  COTS include files.
 */

#include <stdio.h>

/*
 *  Custom include files.
 */

#include "symbol.h"
#include "gcb_parse.h"

/*
 *  Local defines.
 */

#define MAX_ERROR           1024
#define MAX_PARSE_DATA      1024

extern char *yytext;
extern char parse_error_message[];
extern char *parse_data_ptr;
extern int parse_error_value;


yywrap()

{
    parse_error_value = END_OF_FILE;

    return 1;
}


/*
 *  Function yyerror is called from the YACC generated source code when an error is
 *  encountered during the parsing process.
 */

yyerror( err_message )
char *err_message;
{
    if (parse_error_value != END_OF_FILE)
        {
        sprintf(parse_error_message,"%sSyntax error in expression\n",parse_error_message)
;
        parse_error_value = SYNTAX_ERR;
        }
}  /*  of function  */
```

```
/*
 *  Function parse_expression is invoked when an expression is to be parsed.  The functi
on
 *  is passed a character string pointer which points to the expression to be parsed.  T
he
 *  function returns the following values:
 *
 *      PARSE_SUCCESS            (00):      Success
 *      NO_PARSE_MEMORY  (01):      Insufficient storage for parse string
 *      UNDECLARED       (02):      Indicates an undeclared variable
 *      SYNTAX_ERR       (04):      Indicates a syntax error in the expression
 *
 *  the global variable "parse_error_message" can be accessed for more informative human
 *  interpretable diagnostics.
 */

int parse_expression( expression )
char *expression;
{
    char local_data[ MAX_PARSE_DATA ];
    int  i;

    if (expression)
        {

        /*
         *  Initialize the error variables.
         */

        parse_error_value = PARSE_SUCCESS;
        parse_error_message[ 0 ] = NULL;
        stable_state = 0;


        /*
         *  Copy the data to local storage (because the parsing routine will change the
         *  data) and initialize the data pointer to point to the data.
         */


        if ( ( strlen( expression ) + 1 ) > MAX_PARSE_DATA )
            return( NO_PARSE_MEMORY );

        strcpy( local_data, expression );
        parse_data_ptr = local_data;

        }
    else
        {
        stable_state = 1;
        return END_OF_FILE;
        }

    /*
     *  Invoke the parsing routine and return any diagnostic values.
     */

    paren_count = 0;
    WhereAmI = LHS;
    yyparse();

    if ( (parse_error_value == END_OF_FILE) && (stable_state) )
        parse_error_value = PARSE_SUCCESS;

    return( parse_error_value );
}  /*  of function  */
```

```
    displayer("\n\n");
}
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    call_flow.h
*
* FILE FUNCTION:
*
*   This file contains the structure, variable, and function definitions for the
*   Display Comp Call Flow function.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /home/project/3531/Docu/GCB.spec.doc
*
*
* FILE MODULES:
*
*   N/A
*
*****************************************<----->*****************************************/


/*
 *  Call Flow constants.
 */

#define CLEARED         99
#define NOT_CLEARED     100


/*
 *  Linked list node definition and pointer declarations.
 */

struct node
    {
    struct node *child,             /* pointer to first child element          */
                *next;              /* next element pointer (sibling or parent) */
    int         processed;          /* flag used during printing and free()'ing */
    char        name[MAX_NAME],     /* element name                            */
                type;               /* comp or library element                 */
    };


struct node *currPtr,
            *nextPtr,
            *parentPtr,
            *prevPtr,
            *rootPtr;


/*
 *  Function prototypes.
 */

void    free_node_list(),
        indent(),
        print_node_list();
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    cbr.h
*
* FILE FUNCTION:
*
*   This file contains many of the the callback function prototypes for the
*   Graphical Comp Builder.  These prototypes are needed during the building of
*   the MOTIF menus and buttons.
*
*   This file also contains several program constants that relate to MOTIF menus
*   and their current state.
*
*
* FILE MODULES:
*
*   N/A
*
*
*****************************************<----->*****************************************
/


#define CBR_LANG_APPLY  1
#define CBR_LANG_RESET  2
#define CBR_LANG_SELECT 3

#define CBR_COMP_TYPE           0
#define CBR_COMP_TYPE_DONE      1
#define CBR_COMP_TYPE_CANCEL    2


/*
 *  Callback routine function prototypes.
 */


XtCallbackProc  cbr_attribs_done(),
                cbr_audit(),
                cbr_audit_on(),
                cbr_build(),
                cbr_call_flow(),
                cbr_call_tgl(),
                cbr_cancel(),
                cbr_canvas(),
                cbr_clear_popup(),
                cbr_color_choice(),
                cbr_color_done(),
                cbr_color_menu(),
                cbr_color_sym_choice(),
                cbr_copy_element(),
                cbr_create_elem_copy(),
                cbr_cre_comp(),
                cbr_cre_elem(),
                cbr_cre_pos(),
                cbr_done(),
                cbr_elem_popup(),
                cbr_el_delete(),
                cbr_el_selected(),
                cbr_enter_canvas(),
                cbr_exit(),
                cbr_expose(),
                cbr_expr_input(),
                cbr_help(),
                cbr_help_area(),
                cbr_language(),
                cbr_language_menu(),
                cbr_motion_canvas(),
                cbr_new_element(),
                cbr_new_position(),
                cbr_parse(),
                cbr_pause_units(),
                cbr_pause_value(),
                cbr_print(),
                cbr_print_comp(),
                cbr_purpose(),
                cbr_rule_expose(),
                cbr_root_elem(),
                cbr_sav_element(),
                cbr_scroll_expose(),
                cbr_sel_comp(),
                cbr_sel_elem(),
                cbr_sel_pos(),
                cbr_set_sym_display(),
                cbr_show_status(),
                cbr_snap(),
                cbr_symbol(),
                cbr_symbol_move(),
                cbr_text_proc(),
                cbr_tgl_purpose(),
                cbr_validate(),
                cbr_var_type_done();
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    cbr_cancel.c
*
* FILE FUNCTION:
*
*   This file contains the routines which cancel the current or previous operation.
*   This file contains the routines which process the CANCEL button in the Status
*   area.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /home/project/3531/Docu/GCB.spec.doc
*
* FILE MODULES:
*
*   cancel_draw()    - cancels a draw symbol request
*   cbr_cancel()     - overall cancel handler in any mode
*   invert()         - records the last palette item selected
*   popdown()        - takes down the proper popups depending on palette item selected
*
*****************************************<----->*****************************************/

#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>

#include "gcb.h"
#include "constants.h"
#include "widgets.h"
#include "menu.h"
#include "symbol.h"
#include "gcb_parse.h"
#include "lines.h"
#include "cursors.h"

static int  last_chosen;
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cancel_draw()
*
* MODULE FUNCTION:
*
*   This routine handles events when the user cancels a draw operation: it destroys
*   the current draw area widget and pops down the proper window.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  cancel_draw()

{
    int i;

    if ( Mode == AddSymbol )
        {

        /*
         *  Loop through the array of symbol pointers until we find the symbol
         *  the user was working on, then destroy the symbol the user was working
         *  on.  Mark the symbol array pointer as unused.
         */

        for ( i=0; i<MAX_SYMBOLS; i++ )
            if ( symbols[i] == current_symbol )
                {
                /*DEBUG*/
                elog(3,"cancelling draw: unmapping symbol# %i", i);

                XtUnmapWidget( current_symbol );
                XtUnrealizeWidget( current_symbol );
                symbols[i] = NULL;
                break;
                }
        }

    /*
     *  Take down the window, reset to Edit  mode, and update the status indicator
     *  to reflect the change back to Edit mode.
     */

    popdown( last_chosen );
    Mode = EditSymbol;
    upd_mode_panel();
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_cancel()
*
* MODULE FUNCTION:
*
*   This routine handles events when the user pushes the CANCEL button which is
*   found in the status area.  The CANCEL button allows the user to cancel most
*   any operation, including Work Area edits, expression building, etc.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_cancel( w, closure, call_data )

    Widget      w;
    int         closure;
    caddr_t     call_data;

{
    Arg args[1];
    int type;

    if (closure == FILE_OK)
        {

        /*
         *  bring down the help popup; reset the mode if necessary.
         */

        if (Mode == Help)
            {
            Mode = EditSymbol;
            upd_mode_panel();
            }
        XtSetArg( args[0], XmNdialogStyle, XmDIALOG_APPLICATION_MODAL );
        XtSetValues( dlg_file, args, 1 );
        XtUnmanageChild( dlg_file );
        }

    /*
     *  User wants to abort from selecting a User defined function for insertion
     *  into an expression.
     */

    else if (closure == DEF_FN_CANCEL)
        XtUnmanageChild( dlg_def_fn );

    /*
     *  User wants to abort the Help selector/displayer.
     */

    else if (closure == HELP_HELP_CANCEL)
        XtUnmanageChild( dlg_help );

    /*
     *  User wants to abort while in the middle of adding a new symbol.
     */

    else if (Mode == AddSymbol)
        {
        XtSetArg( args[0], XmNuserData, &type );
        XtGetValues( current_symbol, args, 1 );
        if ( (type == IF) || (type == SET) )
            cancel_var_list( expr_text );
        cancel_draw();
        }

    /*
     *  If the matrix functions menu is up, take it down and redisplay the
     *  math/logicals menu.
     */

    else if ( Mode == MatrixMenu )
        {
        XtUnmapWidget( frame_matrix_menu );
        XtMapWidget( frame_math_menu );
        Mode = OldMode;
        }

    /*
     *  If the trigonometric functions menu is up, take it down and redisplay the
     *  math/logicals menu.
     */

    else if ( Mode == TrigMenu )
        {
        XtUnmapWidget( frame_trig_menu );
        XtMapWidget( frame_math_menu );
        Mode = OldMode;
        }

    /*
     *  User wants to abort while in the middle of line draw.
     */

    else if ( Mode == LineDraw )
        {

        /*
         *  Clear the line that is being drawn and restore the mode and cursor, the
         *  cursor is changed to a cross-hair during line draw.
         */

        if ( LDendX != -1 )
            clear_line();
        cancel_line();

        Mode = EditSymbol;

        XDefineCursor( display, XtWindow(draw_area), basic_cursor );
        upd_mode_panel();
        }

    /*
     *  User wants to abort while in the middle of a "right-button" edit.
     */

    else if ( Mode == EditSymbolContents )
        {

        /*
```

```
         *  delete variables introduced during edit; popdown proper window
         */

        XtSetArg( args[0], XmNuserData, &type );
        XtGetValues( current_symbol, args, 1 );
        if ( (type == IF) || (type == SET) )
            cancel_var_list( );

        XtSetArg( args[0], XmNuserData, &type );
        XtGetValues( current_symbol, args, 1 );
        popdown( type );

        Mode = EditSymbol;
        }

    /*
     *  See if the user was in the middle of doing something with a rubber-banded box.
     */

    else if ( ( Mode == StretchPrintBox ) ||
              ( Mode == StretchDeleteBox) ||
              ( Mode == StretchCopyBox)   ||
              ( Mode == StretchMoveBox)   ||
              ( Mode == PrintBox )        ||
              ( Mode == CopyBox )         ||
              ( Mode == MoveBox )         ||
              ( Mode == GhostCopyBox )    ||
              ( Mode == GhostMoveBox )    ||
              ( Mode == DeleteBox) )
            {

            /*
             *  erase box drawn so far; restore mode and cursor
             */

            if ( ( Mode == StretchPrintBox ) ||
                 ( Mode == StretchCopyBox )  ||
                 ( Mode == StretchMoveBox )  ||
                 ( Mode == GhostCopyBox )    ||
                 ( Mode == GhostMoveBox )    ||
                 ( Mode == StretchDeleteBox) )
                draw_ghost( FALSE, BStartx, BStarty, 0, 0 );

            Mode = EditSymbol;
            upd_mode_panel();
            XDefineCursor( display, XtWindow(draw_area), basic_cursor );
            }

    /*
     *  For all other CANCEL possibilities, call undo to undo the previous operation.
     */

    else
        undo();
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  invert()
*
* MODULE FUNCTION:
*
*   This routine records the most recently selected palette item.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  invert( item )

    int item;

{
    last_chosen = item;
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  popdown()
*
* MODULE FUNCTION:
*
*   This routine unmanages the popup corresponding to the parameter palette item type.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  popdown( symbol )

    int symbol;

{
    switch( symbol )
        {

        /*
         *  depending on the type of popup, reset the popup's text fields and
         *  bring down the popup.
         */

        case BEGIN:
        case END:
            break;

        case IF:
        case SET:
            XmTextSetString( scr_logic,   NULLS );
            XmTextSetString( scr_expr,    NULLS );
            XmTextSetString( scr_comment, NULLS );
            XtUnmapWidget( frame_math_menu );
            XtMapWidget( frame_palette );
            XtUnmanageChild( dlg_logic );
            break;

        case GOTO:
            XmTextSetString( txt_call_ne, NULLS );
            XtUnmanageChild( dlg_call );
            break;

        case START:
        case STOP:
            XmTextSetString( txt_start, NULLS );
            XtUnmanageChild( dlg_start );
            break;

        case PAUSE:
            XmTextSetString( txt_pause_value, NULLS );
            XtUnmanageChild( dlg_pause );
            break;

        case TEXT:
        case PRINT:
            XmTextSetString( scr_text, NULLS );
            XtUnmanageChild( dlg_text );
            break;
        }
}
```

```
/*****************************************<----->*****************************************
*
*
* FILE NAME:    cbr_canvas.c
*
* FILE FUNCTION:
*
*   This file contains the routines which  respond to events in the draw area.
*
*
* FILE MODULES:
*
*   cbr_canvas()              - handles mouse button events in the work area canvas.
*   cbr_enter_canvas()        - handles enter events in the drawing area.
*   cbr_expose()              - handles events when the canvas draw area is exposed.
*   cbr_motion_canvas()       - handles motion events in the drawing area.
*   cbr_rule_expose()         - handles events when the ruler bars are exposed.
*   cbr_scroll_expose()       - handles expose events in the windows containing rules.
*   check_from_lines()        - do moved lines entering symbol impose on line or symbol.
*   check_line()              - does single moved line impose on another line or symbol.
*   check_sym()               - does a moved symbol impose on a line or another symbol.
*   copy_box()                - checks impending copy; then copies the symbols in box
*   copy_symbol()             - copies a single symbol within drawn box to new location.
*   copy_symbols()            - copies all the symbols within box to their new locations.
*   copy_text_fields()        - mallocs space for global text vars; copies text fields.
*   draw_ghost()              - erases the old ghosted box and draws a new one.
*   get_copy_from_source()    - determines the new version of a symbol
*   handle_button_press_canvas() - handles button press events in the canvas.
*   line_enters_box()         - determines if a line ever enters the parameter box.
*   line_within_box()         - determines if a line is wholely within the parameter box.
*   move_box()                - determines if box can be moved; then moves contents
*   move_line()               - moves each line segment in a line to its new location.
*   move_lines()              - moves all lines wholely within a box to new locations
*   move_ok()                 - performs move of symbols and lines if move can be done
*   set_up_line_globals()     - sets the globals required for a delete line operation.
*   sym_in_orig_box()         - determines if a symbol was in the original drawn box.
*   sym_within_box()          - determines if a symbol is wholely within parameter box.
*   trial_move()              - determines if the symbols and lines within box, if moved,
*                               will impose on anything in their new locations.
*
*****************************************<----->*****************************************/

#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>

#include "gcb.h"
#include "cbr.h"
#include "widgets.h"
#include "lines.h"
#include "menu.h"
#include "element_file.h"
#include "fonts.h"
#include "cursors.h"
#include "cbr_canvas.h"
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_canvas()
*
* MODULE FUNCTION:
*
*   This routine handles events when the user pushes a mouse button while in
*   the work area canvas.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_canvas( w, client_data, call_data )

    Widget                          w;
    caddr_t                         client_data;
    XmDrawingAreaCallbackStruct     *call_data;

{
    XEvent              *event;
    extern int          Red;
    void                copy_box(), move_box();


/*DEBUG*/
    if ((call_data == NULL) || (call_data->event == NULL))
        {
        printf("\nSomehow got into cbr_canvas with NULL ptr\n");
        return;
        }

    event  = (XEvent *) call_data->event;

    if ( !ValidElement )
        {
        user_ack("A valid Element has not been selected or created");
        return;
        }

    if ( event->type == ButtonPress )
        handle_button_press_canvas( event );

    else if ( event->type == ButtonRelease )
        {
        if ( (event->xbutton.button == Button2) && (Mode == LineDraw) )
            {
            /*
             *  The line has already been drawn, so make sure the line is
             *  legal.  If not, remove the line, let the user know he/she
             *  is stupid, and then let them draw a new line.  Otherwise,
             *  set a new anchor point.
             */

            if ( valid_line(LDtype) == ERR )
                {
                XBell( display, 0 );
                user_ack("Invalid connector - cannot go through a symbol");
                clear_line();
```

```
        }
    else
        {
        set_line();
        mid_Line();
        LDstartX = LDendX;
        LDstartY = LDendY;
        LDtype = MID_SEGMENT;
        }

    LDendX    -1;
    }
else if ( (event->xbutton.button == Button1) && ( (Mode == PrintBox) ||
    (Mode == DeleteBox) ||  (Mode == MoveBox) ||  (Mode == CopyBox)) )
    {

    /*
     *  set ulcx of bounding box and stretch mode; change cursor
     */

    BStartx = event->xbutton.x;
    BStarty = event->xbutton.y;
    if ( Mode == PrintBox )
        Mode = StretchPrintBox;
    else if ( Mode == DeleteBox )
        Mode = StretchDeleteBox;
    else if ( Mode == CopyBox )
        Mode = StretchCopyBox;
    else if ( Mode == MoveBox )
        Mode = StretchMoveBox;
    XDefineCursor( display, XtWindow(draw_area), lr_cursor );

    /*
     *  draw each ghost using Red foreground.
     */

    XSetForeground( display, LDgc, colors[Red] ^ LDbackground );
    }
else if ( (event->xbutton.button == Button1) && ( (Mode == StretchPrintBox) ||
    (Mode == StretchDeleteBox) ||  (Mode == StretchCopyBox)  ||
    (Mode == StretchMoveBox) ) )
    {

    XDefineCursor( display, XtWindow(draw_area), basic_cursor );

    if ( Mode == StretchDeleteBox )
        {
        /*
         *  erase the last banded box
         */

        highlight_box( BStartx, BStarty, old_lrcx, old_lrcy );
        delete_box( BStartx, BStarty, old_lrcx, old_lrcy, TRUE );
        XSetForeground( display, LDgc, colors[Red] ^ LDbackground );
        draw_ghost( FALSE, BStartx, BStarty, 0, 0 );
        }
    else if ( Mode == StretchPrintBox )

        /*
         *  ask the user for print specs
         */

        XtManageChild(dlg_print_elem);
    else if ( (Mode == StretchCopyBox) || (Mode == StretchMoveBox) )
        {
        /*
         *  erase the last banded box
         */

        draw_ghost( FALSE, BStartx, BStarty, 0, 0 );
        ghost_width  = event->xbutton.x - BStartx;
        ghost_length  = event->xbutton.y - BStarty;
        draw_ghost( TRUE, event->xbutton.x, event->xbutton.y,
            event->xbutton.x + ghost_width,  event->xbutton.y + ghost_length );
        }
    if ( Mode == StretchCopyBox )
        Mode = GhostCopyBox;
    else if ( Mode == StretchMoveBox )
        Mode = GhostMoveBox;
    else Mode = EditSymbol;
    upd_mode_panel();
    }
else if ( (event->xbutton.button == Button1) && ( (Mode == GhostCopyBox) ||
    (Mode == GhostMoveBox) ) )
    {

    /*
     *  erase ghost box
     */

    draw_ghost( FALSE, event->xbutton.x, event->xbutton.y,
        event->xbutton.x + ghost_width,  event->xbutton.y + ghost_length );

    /*
     *  copy contents of box from original ulc + size to current ulc + size.
     */

    if ( Mode == GhostCopyBox )
        copy_box( BStartx, BStarty, ghost_width, ghost_length,
            event->xbutton.x, event->xbutton.y );
    else if ( Mode == GhostMoveBox )
        move_box( BStartx, BStarty, ghost_width, ghost_length,
            event->xbutton.x, event->xbutton.y );

    /*
     *  restore default mode.
     */

    Mode = EditSymbol;
    upd_mode_panel();
    }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_enter_canvas()
*
* MODULE FUNCTION:
*
*   This routine handles enter events in the drawing area.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_enter_canvas( w, client_data, event )

    Widget                          w;
    caddr_t                         client_data;
    XCrossingEvent                  *event;

{
    XWindowAttributes   attribs;

    if ( (w == NULL) || (event == NULL) )
        {
printf("cbr_enter_canvas: window or event is null\n");
        return;
        }

    if ( ( event->type == EnterNotify ) && ( Mode == AddSymbol) &&
        (!XtIsManaged(dlg_pause)) && (!XtIsManaged(dlg_call)) &&
        (!XtIsManaged(dlg_logic)) && (!XtIsManaged(dlg_text)) &&
        (!XtIsManaged(dlg_start)) )
        {

        /*
         *  get window x and y
         */

        if (! XGetWindowAttributes(display,XtWindow(current_symbol),&attribs) )
            error_handler( ERR_SYM_ATTRIBS, "cbr_enter_canvas" );

        /*
         *  record midpt and ulc of symbol.
         */

        originx = attribs.width / 2;
        originy = attribs.height / 2;
        newx = attribs.x;
        newy = attribs.y;

        /*
         *  move and map window
         */

        XMoveWindow( display, XtWindow(current_symbol), event->x, event->y );
        XtMapWidget( current_symbol );
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_motion_canvas()
*
* MODULE FUNCTION:
*
*   This routine handles motion events in the drawing area.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_motion_canvas( w, client_data, event )

    Widget                          w;
    caddr_t                         client_data;
    XMotionEvent                    *event;

{
    XWindowAttributes   attribs;

    if ( (w == NULL) || (event == NULL) )
        {
printf("cbr_enter_canvas: window or event is null\n");
        return;
        }

    /*
     *  accept only motion events.
     */

    if ( event->type != MotionNotify )
        return;

    /*
     *  only interested in motion events with no buttons down.
     */

    if ( event->state & Button1Mask)
        elog(3,"motion: button 1 down");
    else if ( event->state & Button2Mask)
        elog(3,"motion: button 2 down");
    else if ( event->state & Button3Mask)
        elog(3,"motion: button 3 down");
    else
        {

        /*
         *  no buttons down; the user is drawing a line.
         */

        if ( Mode == LineDraw )
            {

            /*
             *  Clear previous line if one existed.
             */

            if ( LDendX != -1 )
                clear_line();
            /*
             *  Get current mouse location and draw line from anchored start
             *  point to current mouse location.
             */

            LDendX = event->x;
            LDendY = event->y;

            /*
             *  Determine othogonal direction.
             */

            if ( abs(LDendX-LDstartX) > abs(LDendY-LDstartY) )
                LDendY = LDstartY;
            else
                LDendX = LDstartX;

            draw_line();

            /*
             *  Update the ruler bars.
             */

            XMoveWindow( display, XtWindow(v_rule),  event->x + 2, 0 );
            XMoveWindow( display, XtWindow(h_rule), 0,  event->y + 16 );
            }
        else if ( (Mode == StretchPrintBox) || (Mode == StretchDeleteBox) ||
                  (Mode == StretchCopyBox)  || (Mode == StretchMoveBox) )
            draw_ghost( TRUE, BStartx, BStarty, event->x, event->y );
        else if ( (Mode == GhostCopyBox) || (Mode == GhostMoveBox)  )
            {
            draw_ghost( TRUE, event->x, event->y,
                event->x + ghost_width,  event->y + ghost_length );
            }
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_expose()
*
* MODULE FUNCTION:
*
*   This routine handles events when the canvas draw area is exposed.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_expose( w, client_data, event )

    Widget  w;
    caddr_t client_data;
    XEvent  *event;

{
    XSetWindowAttributes    attribs;

    if ( (w == NULL) || (event == NULL) )
        {
printf("cbr_enter_canvas: window or event is null\n");
        return;
        }

    attribs.backing_store = Always;
    XChangeWindowAttributes( display, XtWindow(draw_area), CWBackingStore, &attribs);

    XUngrabButton( display, AnyButton, AnyModifier, XtWindow(draw_area) );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_rule_expose()
*
* MODULE FUNCTION:
*
*   This routine handles events when the ruler bars are exposed.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_rule_expose( w, client_data, event )

    Widget      w;
    caddr_t     client_data;
    XEvent      *event;

{
    if ( (w == NULL) || (event == NULL) )
        {
printf("cbr_enter_canvas: window or event is null\n");
        return;
        }

    /*
     *   when the ruler bars are exposed, redraw them.
     */

    XClearWindow( display, XtWindow(h_rule) );
    XClearWindow( display, XtWindow(v_rule) );
    XSetFunction( display, WAgc, GXxor );
    XSetForeground( display, WAgc, colors[MAX_COLORS-1] ^ LDbackground );
    XDrawLine( display, XtWindow(h_rule), WAgc, 0, 0, 13, 0 );
    XDrawLine( display, XtWindow(v_rule), WAgc, 0, 0, 0, 13 );
    XSetForeground( display, WAgc, BlackPixel(display, DefaultScreen(display)) ^
        LDbackground );
    XSetFunction( display, WAgc, GXcopy );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_scroll_expose()
*
* MODULE FUNCTION:
*
*   This routine handles events when the windows containing the rules are exposed.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_scroll_expose( w, client_data, event )

    Widget      w;
    caddr_t     client_data;
    XEvent      *event;

{
    int     i;

    /*
     *   when the ruler bars are exposed, redraw them.
     */

    XSetFunction( display, WAgc, GXset );
    XSetForeground( display, WAgc, BlackPixel(display, DefaultScreen(display)) );
    for ( i=1; i < ((MAIN_CANVAS_HEIGHT)/40); i++ )
        XDrawLine( display, XtWindow(v_scroll), WAgc, 0, i*40, 13, i*40 );
    for ( i=1; i < ((MAIN_CANVAS_WIDTH)/40); i++ )
        XDrawLine( display, XtWindow(h_scroll), WAgc, i*40, 0, i*40, 13 );
    XSetFunction( display, WAgc, GXcopy );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  check_from_lines()
*
* MODULE FUNCTION:
*
*   This routine checks the lines entering a symbol to see if, in their new location,
*   they impose on another line or a symbol.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  check_from_lines( type, new_x, new_y, sym, orig_x, orig_y, width, length )

    int     type, new_x, new_y;
    Symbol  *sym;
    int     orig_x, orig_y, width, length;

{
    Line        *templine;
    LineList    *templist;
    LineSeg     *tempseg;
    int         fail;

    /*
     *  do symbol's lines in their new location impose on anything else
     */

    templist = sym->from;
    while ( templist )
        {
        templine = (Line *)templist->line;
        if ( line_within_box(templine, orig_x, orig_y, width, length) )
            {

            /*
             *  trial move line to new location and check it.
             */

            move_line( templine->line, orig_x, orig_y, new_x, new_y );
            fail = check_line( type, templine, orig_x, orig_y, width, length );
            move_line( templine->line, new_x, new_y, orig_x, orig_y );
            if ( fail )
                return( 0 );
            }
        templist = (LineList *)templist->next;
        }
    return( 1 );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  check_line()
*
* MODULE FUNCTION:
*
*   This routine checks a single line to see if, in its new location,
*   it imposes on another line or a symbol.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  check_line( type, line, orig_x, orig_y, width, length )

    int     type;
    Line    *line;
    int     orig_x, orig_y, width, length;

{
    LineSeg     *tempseg;
    int         fail;

    tempseg = (LineSeg *)line->line;
    while ( tempseg )
        {

        /*
         *  check the rectangle enclosed by the line; determine the rectangle based
         *  on the direction of the line.
         */

        if ( tempseg->orientation == RIGHT )
                fail = check_rect( type, tempseg->cell_start_x,
                    tempseg->cell_start_y, tempseg->cell_start_x -
                    tempseg->cell_end_x, 1, orig_x, orig_y, width, length );
        else if ( tempseg->orientation == LEFT )
                 fail = check_rect( type, tempseg->cell_end_x, tempseg->cell_end_y,
                    tempseg->cell_start_x-tempseg->cell_end_x, 1,
                    orig_x, orig_y, width, length );
        else if ( tempseg->orientation == UP )
                 fail = check_rect( type, tempseg->cell_end_x, tempseg->cell_end_y,
                    1, tempseg->cell_start_y-tempseg->cell_end_y,
                    orig_x, orig_y, width, length );
        else if ( tempseg->orientation == DOWN )
                fail = check_rect( type, tempseg->cell_start_x,
                    tempseg->cell_start_y, 1, tempseg->cell_end_y -
                    tempseg->cell_start_y, orig_x, orig_y, width, length );
        if ( !fail )
            return( ERR );
        tempseg = (LineSeg *)tempseg->next;
        }
    return( OK );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  check_sym()
*
* MODULE FUNCTION:
*
*   This routine checks a single symbol to see if, in its new location,
*   it imposes on a line or another symbol.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  check_sym( type, new_x, new_y, sym, orig_x, orig_y, width, length )

    int     type, new_x, new_y;
    Symbol  *sym;
    int     orig_x, orig_y, width, length;

{
    /*
     *  does symbol in its new location impose on anything else
     */

    int cellx = (new_x + (sym->ulcx - orig_x)) / CELL_SIZE;
    int celly = (new_y + (sym->ulcy - orig_y)) / CELL_SIZE;

    /*
     *  can't copy BEGIN symbol
     */

    if ( (sym->symbol_type == BEGIN) && (type == CopyBox) )
        {
        user_ack("can't copy BEGIN symbol");
        return( 0 );
        }

    if ( !check_rect(type, cellx, celly,
        sym->cell_width, sym->cell_height,
        orig_x, orig_y, width, length) )
            return( 0 );

    return( 1 );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  copy_box()
*
* MODULE FUNCTION:
*
*   This routine checks the impending copy for correctness,  then copies all the
*   symbols in the drawn box to their new locations.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  copy_box( orig_x, orig_y, width, length, new_x, new_y )

    int     orig_x, orig_y, width, length, new_x, new_y;

{
    int         numsyms, i;

    /*
     *  reset the copy array.
     */

    for ( i=0; i<MAX_SYMBOLS; i++ )
        {
        copy_array[i].source_num = -1;
        copy_array[i].copy_num = -1;
        }

    /*
     *  if we can copy all the symbols in the box, and no symbols or lines
     *  interfere, create copy of each symbol and line wholly within box.
     */

    if ( numsyms = trial_move(CopyBox, orig_x, orig_y, width, length, new_x, new_y) )
        {

        /*
         *  copy all symbols within box first
         */

        if ( copy_symbols(numsyms,orig_x,orig_y,new_x,new_y) )
            return;

        /*
         *  copy all lines within box
         */

        copy_lines( numsyms, orig_x, orig_y, width, length, new_x, new_y );

        set_null_undo_event();
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  copy_text_fields()
*
* MODULE FUNCTION:
*
*   This routine mallocs space for global text variables; copies text fields.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  copy_text_fields( symbol )

    Symbol  *symbol;

{
    if ( symbol->symbol_type == END )

    /*
     *  no text to copy
     */

        return;

    else if ( (symbol->symbol_type == IF) || (symbol->symbol_type == SET) )
        {

    /*
     *  malloc data space for global text fields depending on symbol's type;
     *  set global text fields to this symbol's text(s)
     */

        if ( symbol->Sym.IfSym.logical_expr )
            {
            if ( (sym_text = (char *)malloc(strlen(symbol->Sym.IfSym.logical_expr) + 1))
                == NULL )
                    {
                    user_ack("Fatal error - Out of memory - couldn't malloc()");
                    elog(1,"copy_symbol(): couldn't malloc() IF text");
                    return( ERR );
                    }
            strcpy( sym_text, symbol->Sym.IfSym.logical_expr );
            }
        else sym_text = NULL;

        if ( symbol->Sym.IfSym.comp_expr )
            {
            if ( (expr_text = (char *)malloc(strlen(symbol->Sym.IfSym.comp_expr) + 1) )
                == NULL )
                    {
                    user_ack("Fatal error - Out of memory - couldn't malloc()");
                    elog(1,"copy_symbol(): couldn't malloc() IF text");
                    return( ERR );
                    }
            strcpy( expr_text, symbol->Sym.IfSym.comp_expr );
            }
        else expr_text = NULL;
```

```
        if ( symbol->Sym.IfSym.comment )
            {
            if ( (comment_text = (char *)malloc(strlen(symbol->Sym.IfSym.comment) + 1) )

                == NULL )
                    {
                    user_ack("Fatal error - Out of memory - couldn't malloc()");
                    elog(1,"copy_symbol(): couldn't malloc() IF text");
                    return( ERR );
                    }
            strcpy( comment_text, symbol->Sym.IfSym.comment );
            }
        else comment_text = NULL;

        }

    else

    /*
     *  goto, start, stop, etc. : just copy the text field
     */

        {
        if ( (sym_text = (char *)malloc(strlen(symbol->text) + 1) ) == NULL )
            {
            user_ack("Fatal error - Out of memory - couldn't malloc()");
            elog(1,"copy_symbol(): couldn't malloc() IF text");
            return( ERR );
            }
        strcpy( sym_text, symbol->text );
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  copy_lines()
*
* MODULE FUNCTION:
*
*   This routine copies all the lines within a drawn box to their new locations.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  copy_lines( numsyms, orig_x, orig_y, width, length, new_x, new_y )

    int  numsyms, orig_x, orig_y, width, length, new_x, new_y;

{
    int         i, from_num, to_num, type;
    Line        *copyline, *templine;
    LineList    *templist;
    Symbol      *tempsym;

    for ( i=0; i<numsyms; i++ )
        {
        templist = Symbol_Map[new_sym_list[i]].from;
        while ( templist )
            {
            templine = (Line *)templist->line;
            if ( line_within_box(templine, orig_x, orig_y, width, length) )
                {

                /*
                 *  copy line; move it; place copy of original.
                 */

                copyline = (Line *)copy_line( templine );
                move_line( copyline->line, orig_x, orig_y, new_x, new_y );
                to_num = get_copy_from_source( new_sym_list[i] );
                from_num = get_copy_from_source( compute_label_index(templine->from) );
                if ( (from_num == -1) || (to_num == -1) )
                    {
                    user_ack("couldnt find copy of sym");
                    return;
                    }
                tempsym = (Symbol *)templine->from;
                if ( tempsym->symbol_type == IF )
                    {
                    if ( tempsym->Sym.IfSym.true_line == templine )
                        type = 1;
                    else if ( tempsym->Sym.IfSym.false_line == templine )
                        type = 2;
                    else user_ack("sym is if but templine != lines");
                    }
                else
                    type = 0;
                restore_line( copyline->line, type, Symbol_Map[from_num].mycanvas,
                    Symbol_Map[to_num].mycanvas );
                }
            templist    (LineList *)templist->next;
```

```
}
}
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  copy_symbols()
*
* MODULE FUNCTION:
*
*   This routine copies all the symbols within a drawn box to their new locations.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  copy_symbols( numsyms, orig_x, orig_y, new_x, new_y )

    int  orig_x, orig_y, new_x, new_y;

{
    int i,
        new_num;

    /*
     *  Record the link between original and copy so that when we are
     *  drawing the lines of the copy, we can know the type of the original's
     *  source symbols.
     */

    for ( i=0; i<numsyms; i++ )
        {

        /*
         *  if copying of symbol fails, return error.
         */

        if ( (new_num = copy_symbol(&Symbol_Map[new_sym_list[i]],
            orig_x, orig_y, new_x, new_y)) == ERR )
                return( ERR );
        copy_array[i].copy_num   = new_num;
        copy_array[i].source_num = new_sym_list[i];
        }
    return( OK );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  copy_symbol()
*
* MODULE FUNCTION:
*
*   This routine copies a single symbol within a drawn box to its new location.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  copy_symbol( symbol, box_x, box_y, new_x, new_y )

    Symbol  *symbol;
    int     box_x, box_y,
            new_x, new_y;

{
    Arg     args[5];
    int     n,
            symx, symy;

    if ( (next_avail_sym()) < 0 )
        return( ERR );

    /*
     *  create new symbol in image of original.
     */

    n = 0;
    XtSetArg( args[n], XmNheight,   symbol->height );      n++;
    XtSetArg( args[n], XmNwidth,    symbol->width );       n++;
    XtSetArg( args[n], XmNuserData, symbol->symbol_type ); n++;
    XtSetValues( current_symbol, args, n );

    /*
     *  move ulc of window to new location relative to copy box
     */

    symx = new_x + ( symbol->ulcx - box_x );
    symy = new_y + ( symbol->ulcy - box_y );

    XMoveWindow( display, XtWindow(current_symbol), symx, symy );

    /*
     *  copy text fields of original sym to global text variables used in
     *  draw_symbol.
     */

    copy_text_fields( symbol );

    /*
     *  draw and install new symbol
     */

    draw_symbol( symbol->symbol_type, current_symbol, WAgc, small_font );
    XtMapWidget( current_symbol );

    if ( (n = install_symbol(symbol->symbol_type, symx, symy, symbol->height,
        symbol->width, sym_text, current_symbol, symbol->font)) >= 0 )
            return( n );
    else
        {
        cancel_draw();
        return( ERR );
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  draw_ghost()
*
* MODULE FUNCTION:
*
*   This routine erases the old ghosted box and draws a new one.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  draw_ghost( flag, ulcx, ulcy, lrcx, lrcy )

    int flag, ulcx, ulcy, lrcx, lrcy;

{
    int i;

    for ( i = 0; (flag==TRUE) ? (i < 2) : (i < 1); i++ )
        {
        XSetFunction( display, LDgc, GXxor );

        /*
         *  draw over old box; if old_lrc? not yet given value, draw box of width
         *  0. Draw new box if flag is true. This enables cancel and print to
         *  erase the last box drawn without drawing a new one by passing in a
         *  false flag.
         */

        XDrawRectangle( display, XtWindow(draw_area), LDgc,
            (i==0) ? old_ulcx : ulcx,
            (i==0) ? old_ulcy : ulcy,
            (i==0) ? ( (old_lrcx >=0) ? abs(old_lrcx-old_ulcx) : 0 )  : abs(lrcx-ulcx),
            (i==0) ? ( (old_lrcy >=0) ? abs(old_lrcy-old_ulcy):0 )  : abs(lrcy-ulcy) );
        if ( i )
            {

            /*
             *  remember the endpts of the box just drawn so we can erase it.
             */

            old_lrcx = lrcx;
            old_lrcy = lrcy;
            old_ulcx = ulcx;
            old_ulcy = ulcy;
            }
        }

        /*
         *  if we are erasing the last ghost box, reset global vars to original values.
         */

        if ( !flag )
            {
            XSetForeground( display, LDgc, BlackPixel(display,
                DefaultScreen(display)) ^ LDbackground );
            old_lrcx = old_lrcy = -1;
            old_ulcx = old_ulcy = 0;
```

```
    }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  get_copy_from_source()
*
* MODULE FUNCTION:
*
*   This routine determines the new version of a symbol by consulting an array that
*   maintains original/copy pairs of symbols.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  get_copy_from_source( source )

{
    int i;

    for ( i=0; i<MAX_SYMBOLS; i++ )
        if ( copy_array[i].source_num == source )
            return( copy_array[i].copy_num );
    return( ERR );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  handle_button_press_canvas()
*
* MODULE FUNCTION:
*
*   As the self-documenting name implies, this routine handles button press events.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  handle_button_press_canvas( event )

    XEvent  *event;

{
    /*DEBUG*/
    elog(3,"cbr_canvas: button #%u down", event->xbutton.button);

    if  ( (Mode == EditSymbol)  && (event->xbutton.button == Button2) &&
        (event->xbutton.state & ShiftMask) )                /*shift down*/

        /*
         *   user wants to delete line.
         */

        {
        if ( !set_up_line_globals(event) )
            return;

        /*
         *   Save seg ptr for undo
         */

        set_line_event( Cell_Map[LDlineY][LDlineX].cell_entry.lines->line );
        delete_line();
        if ( Audit )
            audit( JUST_LINES );
        }
    else if  ( (Mode == EditSymbol)  && (event->xbutton.button == Button1) &&
        (event->xbutton.state & ControlMask) )              /*control down*/
        {

        /*
         *  user wants to audit the line under the cursor.
         */

        if ( !Audited )
            {
            if ( !set_up_line_globals(event) )
                return;
            audit_line();
            Audited = 1;
            }
        else
            {

            /*
```

```
            *  clear any kind of audit by redrawing all lines and symbols.
            */

           clear_audit();
           Audited = 0;
           }
       }
   else if ( (Mode == EditSymbol)  && (event->xbutton.button == Button3) )
       {

       /*
        *  pop up shortcut file menu.
        */

       XmMenuPosition( Rmenu_popup, event );
       XtManageChild( Rmenu_popup );
       }
   else if ( (Mode == EditSymbol)  && (event->xbutton.button == Button1) )
       {

       /*
        *  pop up shortcut edit menu.
        */

       XmMenuPosition( Lmenu_popup, event );
       XtManageChild( Lmenu_popup );
       }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  line_enters_box()
*
* MODULE FUNCTION:
*
*   This routine determines if the parameter line ever enters the parameter box.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  line_enters_box( line, b_ulcx, b_ulcy, w, l )

    Line    *line;
    int     b_ulcx, b_ulcy, w, l;

{
    LineSeg *tempseg;

    tempseg = line->line;
    while ( tempseg )
        {

        /*
         *  if the startpt of the seg is within the box, return true
         */

        if ( ((tempseg->start_x > b_ulcx) && (tempseg->start_x < b_ulcx + w) ) &&
            ( (tempseg->start_y > b_ulcy) && (tempseg->start_y < b_ulcy + l)) )
                return( 1 );

        /*
         *  if the endpt of the seg is within the box, return true
         */

        else if ( ((tempseg->end_x > b_ulcx) && (tempseg->end_x < b_ulcx + w) ) &&
            ( (tempseg->end_y > b_ulcy) && (tempseg->end_y < b_ulcy + l)) )
                return( 1 );

        else tempseg = (LineSeg *)tempseg->next;
        }
    return( 0 );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  line_within_box()
*
* MODULE FUNCTION:
*
*   This routine determines if the parameter line is wholely within the parameter box.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  line_within_box( line, b_ulcx, b_ulcy, w, l )

    Line    *line;
    int     b_ulcx, b_ulcy, w, l;

{
    LineSeg *tempseg;

    /*
     *  if line->line->start_[xy] in box...
     */

    tempseg = line->line;
    while (tempseg)
        {
        if ( !( ( (tempseg->start_x > b_ulcx) && (tempseg->start_x < b_ulcx + w) ) &&
            ( (tempseg->start_y > b_ulcy) && (tempseg->start_y < b_ulcy + l)) ) )
                return( 0 );

        /*
         *  and if line->line->end_[xy] in box...
         */

        else if ( !( ( (tempseg->end_x > b_ulcx) && (tempseg->end_x < b_ulcx + w) ) &&
            ( (tempseg->end_y > b_ulcy) && (tempseg->end_y < b_ulcy + l)) ) )
                return( 0 );
        else tempseg = (LineSeg *)tempseg->next;
        }
    return( 1 );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  move_box()
*
* MODULE FUNCTION:
*
*   This routine determines if the drawn box can be moved; if so, it calls the routine
*   to perform the move.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  move_box( orig_x, orig_y, width, length, new_x, new_y )

    int     orig_x, orig_y, width, length, new_x, new_y;

{
    int     numsyms;

    /*
     *  try to move each symbol; if we can, move each symbol and each line
     *  entering it that is wholly within the box. Delete all other lines.
     */

    if ( numsyms = trial_move(MoveBox, orig_x, orig_y, width, length, new_x, new_y) )
        {
        move_ok( numsyms, new_x, new_y, orig_x, orig_y, width, length );
        set_null_undo_event();
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  move_labels()
*
* MODULE FUNCTION:
*
*   This routine moves the TRUE/FALSE labels of the parameter symbol to their
*   new locations.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  move_labels( i, new_x, new_y, orig_x, orig_y )

    int     i, new_x, new_y, orig_x, orig_y;

{
    /*
     *  this is not pretty, but since the label info is in the symbol, not the
     *  line, the labels must be moved before the symbol. We know the new
     *  location; just clear the old labels and draw them in their new
     *  location.
     */

    if ( Symbol_Map[new_sym_list[i]].Sym.IfSym.true_line )
        {
        clear_text( draw_area, Symbol_Map[new_sym_list[i]].Sym.IfSym.true_x,
            Symbol_Map[new_sym_list[i]].Sym.IfSym.true_y, "TRUE" );
        Symbol_Map[new_sym_list[i]].Sym.IfSym.true_y =
            new_y + ( Symbol_Map[new_sym_list[i]].Sym.IfSym.true_y - orig_y );
        Symbol_Map[new_sym_list[i]].Sym.IfSym.true_x =
            new_x + ( Symbol_Map[new_sym_list[i]].Sym.IfSym.true_x - orig_x );
         XDrawString( display, XtWindow(draw_area), LDgc,
            Symbol_Map[new_sym_list[i]].Sym.IfSym.true_x,
            Symbol_Map[new_sym_list[i]].Sym.IfSym.true_y,
            "TRUE", strlen("TRUE") );
        }
    if ( Symbol_Map[new_sym_list[i]].Sym.IfSym.false_line )
        {
        clear_text( draw_area, Symbol_Map[new_sym_list[i]].Sym.IfSym.false_x,
            Symbol_Map[new_sym_list[i]].Sym.IfSym.false_y, "FALSE" );
        Symbol_Map[new_sym_list[i]].Sym.IfSym.false_y =
            new_y + ( Symbol_Map[new_sym_list[i]].Sym.IfSym.false_y - orig_y );
        Symbol_Map[new_sym_list[i]].Sym.IfSym.false_x =
            new_x + ( Symbol_Map[new_sym_list[i]].Sym.IfSym.false_x - orig_x );
        XDrawString( display, XtWindow(draw_area), LDgc,
            Symbol_Map[new_sym_list[i]].Sym.IfSym.false_x,
            Symbol_Map[new_sym_list[i]].Sym.IfSym.false_y,
            "FALSE", strlen("FALSE") );
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  move_line()
*
* MODULE FUNCTION:
*
*   This routine moves each line segment in the parameter line to its new location.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  move_line( line, orig_x, orig_y, new_x, new_y )

    LineSeg     *line;
    int         orig_x, orig_y, new_x, new_y;

{
    LineSeg     *tempseg = line;

    while ( tempseg )
        {

        /*
         *  set each pt to new box ulc + offset into original box.
         */

        tempseg->start_x = new_x + ( tempseg->start_x - orig_x );
        tempseg->start_y = new_y + ( tempseg->start_y - orig_y );
        tempseg->end_x = new_x + ( tempseg->end_x - orig_x );
        tempseg->end_y = new_y + ( tempseg->end_y - orig_y );
        tempseg->cell_start_x = tempseg->start_x / CELL_SIZE;
        tempseg->cell_start_y = tempseg->start_y / CELL_SIZE;
        tempseg->cell_end_x = tempseg->end_x / CELL_SIZE;
        tempseg->cell_end_y = tempseg->end_y / CELL_SIZE;
        if ( tempseg->arrow_x )
            tempseg->arrow_x = new_x + ( tempseg->arrow_x - orig_x );
        if ( tempseg->arrow_y )
            tempseg->arrow_y = new_y + ( tempseg->arrow_y - orig_y );
        tempseg = (LineSeg *)tempseg->next;
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  move_lines()
*
* MODULE FUNCTION:
*
*   This routine moves all lines of the parameter symbol number to their new
*   locations.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  move_lines( sym_num, new_x, new_y, orig_x, orig_y, width, length, numsyms )

        int     sym_num, new_x, new_y, orig_x, orig_y, width, length, numsyms;

{
    Line        *templine;
    LineList    *templist, *nextlist;
    LineSeg     *tempseg;
    int         done = -1;

    templist = Symbol_Map[new_sym_list[sym_num]].from;
    while ( templist )
        {

        /*
         *  go through all lines entering this symbol; for each one within
         *  the move box, erase it, free its cells, and redraw it in its new
         *  location.
         */

        nextlist = (LineList *)templist->next;
        templine = (Line *)templist->line;
        if ( (line_within_box(templine, orig_x, orig_y, width, length)) &&
            (sym_within_box(templine->from, orig_x, orig_y, width, length)) )
            {
            erase_line( templine );
            LDlinePtr = (Line *)templine;
            tempseg = (LineSeg *)templine->line;
            while ( tempseg )
                {
                clear_cell_map_line( tempseg );
                tempseg = (LineSeg *)tempseg->next;
                }
            move_line( templine->line, orig_x, orig_y, new_x, new_y );
            draw_whole_line( templine, TRUE );
            }
        else

        /*
         *  line enters symbol but is outside move box; delete line -
         *  set globals first.
         */

            {
            find_line_cell( templine->line );
            delete_line();
            }

        templist = (LineList *)nextlist;
        }
    while ( done < 1 )
        {

        /*
         *  set templine to the next (or true/false) lines of the symbol;
         *  if the line is not within the box, delete it.
         */

        if ( Symbol_Map[new_sym_list[sym_num]].symbol_type == IF )
            {
            if ( done == -1 )
                {
                templine = Symbol_Map[new_sym_list[sym_num]].Sym.IfSym.true_line;
                done = 0;
                }
            else
                {
                templine = Symbol_Map[new_sym_list[sym_num]].Sym.IfSym.false_line;
                done = 1;
                }
            } else
            {
            templine = Symbol_Map[new_sym_list[sym_num]].next;
            done = 1;
            }

        if ( templine )
            {
            if ( !sym_in_orig_box(compute_label_index(templine->to), numsyms) )

                /*
                 *  delete line - set globals first.
                 */

                {
                find_line_cell( templine->line );
                delete_line();
                }
            }
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  move_ok()
*
* MODULE FUNCTION:
*
*   This routine performs the move of symbols and lines once it has been determined
*   that the move can be performed safely.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  move_ok( numsyms, new_x, new_y, orig_x, orig_y, width, length )

    int     numsyms, new_x, new_y, orig_x, orig_y, width, length;

{
    int     i, symx, symy, old_x, old_y;

    for ( i=0; i<numsyms; i++ )
        {

        /*
         *  save old symbol location; test in new location will change these
         *  values.
         */

        old_x = Symbol_Map[new_sym_list[i]].ulcx;
        old_y = Symbol_Map[new_sym_list[i]].ulcy;

        /*
         *  offset into new box location.
         */

        symx = new_x + ( Symbol_Map[new_sym_list[i]].ulcx - orig_x );
        symy = new_y + ( Symbol_Map[new_sym_list[i]].ulcy - orig_y );


        if ( Symbol_Map[new_sym_list[i]].symbol_type == IF )
            move_labels( i, new_x, new_y, orig_x, orig_y );

        /*
         *  move symbol; try to claim cells in new location.
         */

        XMoveWindow( display,XtWindow(Symbol_Map[new_sym_list[i]].mycanvas),symx,symy );
        if ( !claim_cells(Symbol_Map[new_sym_list[i]].mycanvas) )
            {
            user_ack("move_ok: can't claim symbol cells");
            return( 0 );
            }

        /*
         *  restore old symbol location; test in new location will have changed
         *  these values.
         */

        Symbol_Map[new_sym_list[i]].ulcx = old_x;
        Symbol_Map[new_sym_list[i]].ulcy = old_y;

        /*
         *  relocate all lines entering this symbol that are wholly within box;
         *  delete all others.
         */

        move_lines( i, new_x, new_y, orig_x, orig_y, width, length, numsyms );
        }

    /*
     *  set cells for all lines now, so they don't clash with symbols earlier.
     */

    set_cell_map_lines(numsyms);

    for ( i=0; i<numsyms; i++ )
        {
        Symbol_Map[new_sym_list[i]].ulcx = new_x +
            ( Symbol_Map[new_sym_list[i]].ulcx - orig_x );
        Symbol_Map[new_sym_list[i]].ulcy = new_y +
            ( Symbol_Map[new_sym_list[i]].ulcy - orig_y );
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  set_cell_map_lines()
*
* MODULE FUNCTION:
*
*   This routine sets the cells for all the lines entering all the moved symbols.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  set_cell_map_lines( numsyms )

    int  numsyms;

{
    Line        *templine;
    LineList    *templist;
    LineSeg     *tempseg;
    int         i;

    for ( i=0; i<numsyms; i++ )
        {

        /*
         *  for each line entering this symbol, set the cells occupied by
         *  each segment.
         */

        templist = (LineList *)Symbol_Map[new_sym_list[i]].from;
        while ( templist )
            {
            templine = (Line *)templist->line;
            tempseg = (LineSeg *)templine->line;
            while ( tempseg )
                {
                set_cell_map_line( tempseg, templine );
                tempseg = (LineSeg *)tempseg->next;
                }
            templist = (LineList *)templist->next;
            }
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  set_up_line_globals()
*
* MODULE FUNCTION:
*
*   This routine sets the globals required for a delete line operation.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  set_up_line_globals( event )

    XEvent  *event;

{

    /*
     *  Determine if a line exists at this cell.
     */

    LDlineX = event->xbutton.x;
    LDlineY = event->xbutton.y;

    LDlineX = LDlineX / CELL_SIZE;
    LDlineY = LDlineY / CELL_SIZE;

    /*
     *  If this cell is not a line cell, beep at the user.
     */

    if ( Cell_Map[LDlineY][LDlineX].cell_type != LINE_CELL )
        {
        XBell( display, 0 );
        return( 0 );
        }

    /*
     *  If this cell contains multiple lines, let the user
     *  know that we can't tell which line to delete.
     */

    if ( Cell_Map[LDlineY][LDlineX].cell_entry.lines->next )
        {
        user_ack("Cannot determine unique line, please reselect line");
        return( 0 );
        }
    return( 1 );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  sym_in_orig_box()
*
* MODULE FUNCTION:
*
*   This routine determines if the parameter symbol was in the original drawn box.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  sym_in_orig_box( sym_num, numsyms )

    int sym_num, numsyms;

{
    int i;

    for ( i=0; i<numsyms; i++ )
        if ( new_sym_list[i] == sym_num )
            return( 1 );
    return( 0 );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  sym_within_box()
*
* MODULE FUNCTION:
*
*   This routine determines if a symbol is wholely within the parameter box.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  sym_within_box( symbol, b_ulcx, b_ulcy, w, l )

    Symbol  *symbol;
    int     b_ulcx, b_ulcy, w, l;

{
    int lrcx = symbol->ulcx + symbol->width;
    int lrcy = symbol->ulcy + symbol->height;

    /*
     *  if sym->ulcx in box...
     */

    if ( ( (symbol->ulcx > b_ulcx) && (symbol->ulcx < b_ulcx + w) ) &&
        ( (symbol->ulcy > b_ulcy) && (symbol->ulcy < b_ulcy + l) ) )
        {

        /*
         *  if sym->lrcx in box...
         */

        if ( (lrcx < b_ulcx + w) && (lrcy < b_ulcy + l) )
            return( 1 );
        }
    return( 0 );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  trial_move()
*
* MODULE FUNCTION:
*
*   This routine determines if the symbols and lines within a box, if moved, will
*   impose on anything in their new locations.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  trial_move( type, orig_x, orig_y, width, length, new_x, new_y )

    int  type, orig_x, orig_y, width, length, new_x, new_y;

{
    int     j = 0,
            i;

    /*
     *  reset moved symbol list.
     */

    for ( i=0; i<MAX_SYMBOLS; i++ )
        new_sym_list[i] = 0;

    for ( i=0; i<MAX_SYMBOLS; i++ )
        if ( Symbol_Map[i].symbol_type != NONE )
            if ( sym_within_box(&Symbol_Map[i], orig_x, orig_y, width, length) )
                {
                new_sym_list[j] = i;
                j++;

                /*
                 *  does symbol in its new location impose on anything else
                 */

                if ( !check_sym(type, new_x, new_y, &Symbol_Map[i],  orig_x, orig_y,
                    width, length) )
                        return( 0 );

                /*
                 *  do symbol's lines in their new location impose on anything else
                 */

                if ( !check_from_lines(type,new_x,new_y,&Symbol_Map[i],orig_x,orig_y,
                    width, length) )
                        return( 0 );

                }
    return( j );
}
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    cbr_canvas.h
*
* FILE FUNCTION:
*
*   This file contains the global variables and function prototypes for cbr_canvas.c
*
*
* SPECIFICATION DOCUMENTS:
*
*   /home/project/3531/Docu/GCB.spec.doc
*
*
* FILE MODULES:
*
*   N/A
*
*****************************************<----->*****************************************/

/*
 *  This structure establishes a link between an original symbol and its copy.
 *  This link is necessary so that when drawing the lines of the new symbol,
 *  we can determine the type of the original start symbol.
 */

typedef struct
    {
    int source_num;
    int copy_num;
    } copystruct;

copystruct  copy_array[MAX_SYMBOLS];

/*
 *  Global variables.
 */

/*
 *  Record original position of a symbol during a drag.
 */

extern int  originx, originy,
            newx, newy;

int         ghost_width,  ghost_length,     /*  dims of box for move, print, etc.  */
            new_sym_list[MAX_SYMBOLS],

            /*
             *  record dims of last ghosted box so we can erase it.
             */

            old_lrcx = -1, old_lrcy = -1,
            old_ulcx =  0, old_ulcy =  0;

/*
 *  Function prototypes.
 */

void        copy_lines(),
            draw_ghost(),
            move_line(),
            set_cell_map_lines();
```

```
/****************************************<----->****************************************
*
* FILE NAME:    cbr_done.c
*
* FILE FUNCTION:
*
*   This file contains the routines which finish the symbol creation operation.
*   These routines are called when the user selects the DONE button in the
*   symbol specific popups.
*
*
* FILE MODULES:
*
*   cbr_done()              - processes the symbol specific DONE buttons
*   check_lines()           - are all lines entering/leaving symbol still in symbol?
*   check_lineseg()         - if lineseg no longer in resized symbol, delete it.
*   reclaim_line_cells()    - reclaims cells of line leaving bottom of resized symbol
*   redraw_labels()         - redraws labels of IF/SET symbol obscured during resizing
*   redraw_last_lineseg()   - checks lines leaving out bottom of resized IF/SET symbol
*   reinstall()             - changes size, content fields in Symbol_Map entry
*
****************************************<----->****************************************/

#include <stdio.h>
#include <stdlib.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>

#include "gcb.h"
#include "widgets.h"
#include "lines.h"
#include "symbol.h"
#include "gcb_parse.h"
#include "element_file.h"
#include "fonts.h"

#define IN      1
#define OUT     2
```

```
/****************************************<----->****************************************
*
* MODULE NAME:  cbr_done()
*
* MODULE FUNCTION:
*
*   This routine handles events when the user clicks DONE in a symbol specific
*   popup.  This routine determines which type of symbol the user is working
*   on and then acts accordingly.  In the case of SET and IF symbols, the
*   users expression must be checked to ensure it is syntactically correct
*   that all variables are accounted for.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
****************************************<----->****************************************/

XtCallbackProc  cbr_done( w, client_data, call_data )

    Widget  w;
    int     client_data;
    caddr_t call_data;

{
    Symbol  *symbol;
    Arg     args[10];
    char    response[80];
    int     n, rc, sym_type, add;

    /*
     *  if user clicked cancel, return
     */

    if ( client_data == 0 )
        {
        XtManageChild( dlg_logic_attribs );
        return;
        }

    /*
     *  Determine which type of symbol the user is working on.
     */

    XtSetArg( args[0], XmNuserData, &sym_type );
    XtGetValues( current_symbol, args, 1);

    /*DEBUG*/
    elog(1,"cbr_done: client data is %d", sym_type);

    /*
     *  Process the DONE button based on the current symbol's type -> IF, SET,
     *  etc.
     */

    switch( sym_type )
        {
        case START:
        case STOP:

            /*
```

```
         *  Make sure the user entered a Comp name
         */

        if ( !strlen(XmTextGetString(txt_start)) )
            {
            user_ack("Comp name is empty, please be sure to enter a Comp name");
            popdown( sym_type );
            Mode = EditSymbol;
            return;
            }

        /*
         *  Malloc some data space for the string.  The symbol structure
         *  contains a pointer only, we must get data space for the string
         *  at this point.
         */

        if ( (sym_text = malloc(strlen(XmTextGetString(txt_start))+1)) == NULL )
            {
            user_ack("Fatal error - Out of memory - couldn't malloc()");
            elog(1,"cbr_done(): couldn't malloc() START/STOP text");
            return;
            }

        /*
         *  Copy the string from the widget.
         */

        strcpy( sym_text, XmTextGetString(txt_start) );

        /*
         *  Since we are not using expr_text, set it to NULL.
         */

        expr_text = NULL;
        break;

    case GOTO:

        /*
         *  Make sure the user entered an element name
         */

        if ( !strlen(XmTextGetString(txt_call_ne)) )
            {
            user_ack("Element name is empty");
            popdown( sym_type );
            Mode = EditSymbol;
            return;
            }

        /*
         *  Malloc some data space for the string.  The symbol structure
         *  contains a pointer only, we must get data space for the string
         *  at this point.
         */

        if ( (sym_text = malloc(strlen(XmTextGetString(txt_call_ne))+1)) == NULL )
            {
            user_ack("Fatal error - Out of memory - couldn't malloc()");
            elog(1,"cbr_done(): couldn't malloc() GOTO text");
            return;
            }

        /*
         *  Get the element name from the text widget, pick out only the
         *  Element name, dump the trailing "Root Element" text if it
         *  exists.
         */

        sscanf( XmTextGetString(txt_call_ne), "%s", sym_text );

        /*
         *  Determine which type of Element the user selected.
         */

        if ( XmToggleButtonGetState(tgl_call_ce) )
            CallType = ELEMENT;
        else
            CallType = LIB;

        /*
         *  Since we are not using expr_text, set it to NULL.
         */

        expr_text = NULL;
        break;

    case TEXT:
    case PRINT:
        if ( sym_type == PRINT )
            {

            /*
             *  Make sure the user has entered the quotes properly for the
             *  PRINT statement.  There should only be two quotes in the
             *  string.
             */

            rc = count_quotes( XmTextGetString(scr_text) );
            if ( rc != 0 )
                {
                user_ack("String is improperly formed, please remove quotes -> \"");
                return;
                }
            }

        /*
         *  Malloc some data space for the string.  The symbol structure
         *  contains a pointer only, we must get data space for the string
         *  at this point.
         */

        if ( (sym_text = malloc(strlen(XmTextGetString(scr_text)) +1) )
                == NULL)
            {
            user_ack("Fatal error - Out of memory - couldn't malloc()");
            elog(1,"cbr_done(): couldn't malloc() PRINT/TEXT text");
            return;
            }

        /*
         *  Copy the text in the widget to our data space in the symbol
         *  structure.
         */

        strcpy( sym_text, XmTextGetString(scr_text) );
        expr_text = NULL;
```

```
        break;

    case PAUSE:

        if ( !check_pause_string(XmTextGetString(txt_pause_value)) )
            {
            user_ack("Error in Pause string - please reenter Pause symbol");
            return;
            }

        /*
         *  Malloc some data space for the string.  The symbol structure
         *  contains a pointer only, we must get data space for the string
         *  at this point.
         */

        if ( (sym_text = malloc(strlen(XmTextGetString(txt_pause_value)) +1))==NULL)
            {
            user_ack("Fatal error - Out of memory - couldn't malloc()");
            elog(1,"cbr_done(): couldn't malloc() PAUSE text");
            return;
            }

        /*
         *  Copy the text in the widget to our data space in the symbol
         *  structure.
         */

        strcpy( sym_text, XmTextGetString(txt_pause_value) );
        expr_text = NULL;
        break;

    case IF:
    case SET:

        /*
         *  Malloc some data space for the string.  The symbol structure
         *  contains a pointer only, we must get data space for the string
         *  at this point.
         */

        if ( (sym_text = malloc(strlen(XmTextGetString(scr_logic)) +1) )
                == NULL)
            {
            user_ack("Fatal error - Out of memory - couldn't malloc()");
            elog(1,"cbr_done(): couldn't malloc() IF text");
            return;
            }

        /*
         *  Copy the logical text from the widget to our data space.
         */

        if ( strlen(XmTextGetString(scr_logic)) )
            strcpy( sym_text, XmTextGetString(scr_logic) );
        else
            sym_text = NULL;

        /*
         *  Malloc some data space for the string.  The symbol structure
         *  contains a pointer only, we must get data space for the string
         *  at this point.
         */

        if ( (expr_text = malloc(strlen(XmTextGetString(scr_expr)) +1) )
                == NULL)
            {
            user_ack("Fatal error - Out of memory - couldn't malloc()");
            elog(1,"cbr_done(): couldn't malloc() IF text");
            return;
            }

        /*
         *  Copy the logical expresion text from the widget to our data
         *  space.
         */

        if ( strlen(XmTextGetString(scr_expr)) )
            strcpy( expr_text, XmTextGetString(scr_expr) );
        else
            expr_text = NULL;

        /*DEBUG*/
        elog(3,"cbr_done: expr_text: %s", expr_text);

        /*
         *  Malloc some data space for the string.  The symbol structure
         *  contains a pointer only, we must get data space for the string
         *  at this point.
         */

        if ( (comment_text = malloc(strlen(XmTextGetString(scr_comment)) +1))
                == NULL)
            {
            user_ack("Fatal error - Out of memory - couldn't malloc()");
            elog(1,"cbr_done(): couldn't malloc() IF text");
            return;
            }

        /*
         *  Copy the comment text from the widget to our local data space.
         */

        if ( strlen(XmTextGetString(scr_comment)) )
            strcpy( comment_text, XmTextGetString(scr_comment) );
        else
            comment_text = NULL;

        /*DEBUG*/
        elog(3,"cbr_done: comment_text: %s", comment_text);
        break;
    }

/*
 *  We have set the pointers in the symbol to the text strings from
 *  the widget.  Make sure the user entered some text.
 */

/*DEBUG*/
elog(3,"cbr_done: sym_text: %s", sym_text);

if ( (sym_text == NULL) && (expr_text == NULL) )
    {
    user_ack("The expression is emtpy; aborting edit/create");
    elog(1,"cbr_done: no sym or expr text; cancelling");
    popdown( sym_type );
    Mode = EditSymbol;
    return;
```

```
    }

/*
 *  if we are editing a symbol, add new variables to existing var list;
 *  else create new list.
 */

if ( Mode == EditSymbolContents )
    add = 0;
else
    add = 1;

/*DEBUG*/
elog(3,"cbr_done: building new var list for %s", XmTextGetString(scr_expr));

/*
 *  Update the symbol table and verify that the expression is correct.  The
 *  user may have entered an expression via the keyboard so we must check
 *  to make sure it is correct.
 */

if ( (sym_type == IF) || (sym_type == SET) )
    {
    if ( build_new_var_list(XmTextGetString(scr_expr), add) )
        {
        if (sym_type == IF)
            {

            /*
             *  can't have matrix in if expr
             */

            if ( matrix_in_expr(expr_text) != SUCCESS )
                {
                user_ack("IF expressions may not contain matrices");
                return;
                }

            /*
             *  check if expr for syntactic correctness.
             */

            if ( expr_text )
                {
                n =  parse_expression(expr_text);
                if ( (n != PARSE_SUCCESS) && (n != END_OF_FILE) )
                    {
                    user_ack("Syntax error in expression");
                    return;
                    }
                }

            /*
             *  Don't allow strings in IF expressions
             */

            if ( (sym_type == IF) && (string_in_expr(expr_text)) )
                    {
                    user_ack("String not allowed in IF expression");
                    return;
                    }
            }

        /*
         *  check set expr for syntactic and type correctness.
         */

        else if ( verify_expression_type(expr_text, response) != VALID_EXPRESSION )
            {
            /*DEBUG*/
            elog(3,"cbr_done: putting up user ack with str %s", response);
            user_ack( response );
            return;
            }
        else
            /*DEBUG*/
            elog(3,"cbr_done: valid expression");
        }
    else
        return;
    }

/*
 *  the size of 4 of the symbol types is determined by their contents.
 */

if ( (sym_type == IF)    || (sym_type == SET)  ||
     (sym_type == PRINT) || (sym_type == TEXT) )
    {

    /*
     *  Set the symbol size.
     */

    if ( XmToggleButtonGetState(tgl_sym_size_lg) )
        XtSetArg( args[0], XmNwidth, (Zoomed)?op(76 + 120, 2) : 76 + 120);
    else
        XtSetArg( args[0], XmNwidth, (Zoomed)?op(76 + 80, 2) : 76 + 80);

    XtSetValues( current_symbol, args, 1 );

    /*
     *  Set the symbol font.
     */

    if ( XmToggleButtonGetState(tgl_font_size_lg) )
        XSetFont( display, WAgc, WAFont = (Font) big_font );
    else
        XSetFont( display, WAgc, WAFont = (Font) small_font );
    }
else
    XSetFont( display, WAgc, WAFont = (Font) small_font);

/*
 *  Draw the symbol based on the text strings and the attributes (font,
 *  symbol size).
 */

draw_symbol( sym_type, current_symbol, WAgc, WAFont );

/*
 *  if editing symbol contents, try to claim cells for
 *  newly resized symbol and lines entering and leaving it.
 */

if (Mode == EditSymbolContents)
    {
```

```
        /*
         *  retrieve this widget's Symbol_Map entry.
         */

        symbol = (Symbol *) get_sym_map_entry( current_symbol );

        /*
         *  adjust the use counts of this string variables and the old version
         *  of this string.
         */

        if ( (sym_type == IF) || (sym_type == SET) )
            edit_var_list( symbol->Sym.IfSym.comp_expr, XmTextGetString(scr_expr) );

        /*
         *  Try to claim the cells of the (possibly) resized symbol;
         *  if not, revert to previous size and text, else change Symbol_Map
         *  fields for this symbol.
         */

        reinstall();

        Mode = EditSymbol;
        upd_mode_panel();

        /*
         *  adding this symbol may make the element complete; check and update.
         */

        if ( complete(0,LINES_AND_EXPR) == ERR )
            upd_status( 1 );
        else
            upd_status(0);
        }

    /*
     *  Take down the popup now that we are finished with our checks.
     */

    SaveNeeded = True;
    popdown( sym_type );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  check_lines()
*
* MODULE FUNCTION:
*
*   This routine checks all the lines entering and leaving the newly resized symbol
*   to ensure that they still enter or leave the new area size.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  check_lines( symbol )

    Symbol  *symbol;

{
    LineList    *templist,
                *nextlist;
    LineSeg     *tempseg;

    /*
     *  Find line(s) leaving symbol; send to check_lineseg.  The lines leaving
     *  an IF are different pointers than other symbols because there are two
     *  lines: TRUE and FALSE.
     */

    if ( symbol->symbol_type == IF )
        {
        if ( symbol->Sym.IfSym.false_line )
            check_lineseg( symbol->Sym.IfSym.false_line->line, symbol, OUT );
        if ( symbol->Sym.IfSym.true_line )
            check_lineseg( symbol->Sym.IfSym.true_line->line, symbol, OUT );
        }
    else if ( symbol->next )
            check_lineseg( symbol->next->line, symbol, OUT );

    /*
     *  Find line(s) entering symbol; send to check_lineseg.
     */

    if ( symbol->from )
        {
        templist = (LineList *) symbol->from;
        nextlist = (LineList *) templist->next;
        while ( templist )
            {
            tempseg = templist->line->line;

            /*
             *  Get to last seg in line, see if its still within symbol.
             */

            while ( tempseg->next )
                tempseg = (LineSeg *) tempseg->next;

            check_lineseg( tempseg, symbol, IN );
```

```
        templist = (LineList *) nextlist;
        if ( nextlist )
            nextlist = (LineList *) nextlist->next;
        }
    }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  check_lineseg()
*
* MODULE FUNCTION:
*
*   When a symbol shrinks, it is possible that a line that used to enter it at the
*   bottom no longer does. This routine determines this case and deletes the line.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  check_lineseg( lineseg, symbol, direction )

    LineSeg *lineseg;
    Symbol  *symbol;
    int     direction;

{
    switch ( lineseg->orientation )
        {
        case RIGHT:
        case LEFT:
            if ( lineseg->start_y > (symbol->ulcy + symbol->height) )
                {

                /*
                 *  this lineseg no longer enters this symbol; delete the line.
                 */

                if ( direction == IN )
                    {

                    /*
                     *  get back to first seg in line, since that is where
                     *  find_line_cell starts from.
                     */

                    while ( lineseg->prev )
                        lineseg = (LineSeg *) lineseg->prev;
                    }
                find_line_cell( lineseg );
                delete_line();
                }
            break;

        default:

            /*
             *  ignore UP/DOWN lines.
             */

            break;
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  check_pause_string()
*
* MODULE FUNCTION:
*
*   This routine checks the value/units pair produced by the pause popup to ensure
*   that there is exactly 1 value and 1 unit, they are in the correct order, and
*   the unit is a recognized string.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  check_pause_string( str )

    char    *str;

{
    char    value[3], units[4], excess[1];
    int     i, j;

    /*
     *  if >2 args in str, error.
     */

    if ( sscanf(str, "%s %s %s" , value, units, excess) != 2 )
        return( 0 );

    /*
     *  if first string not a number, error.
     */

    j = strlen(value);
    for ( i=0; i<j; i++ )
        if ( !isdigit(value[i]) )
            return( 0 );

    /*
     *  if second string not a valid unit, error.
     */

    if ( (strcmp(units, "ms")) && (strcmp(units, "sec")) && (strcmp(units, "min")) )
        return( 0 );

    return( 1 );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  reclaim_line_cells()
*
* MODULE FUNCTION:
*
*   This routine reclaims the cells of line leaving out the bottom of a newly
*   resized symbol, since they may have become symbol cells during the resizing.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  reclaim_line_cells( line, type )

    Line    *line;
    int     type;   /* 0 for NEXT line, 1 for TRUE line, 2 for FALSE line */

{
    LineSeg  *tempseg = (LineSeg *) line->line;
    Symbol      *from = (Symbol *) line->from;
    Symbol      *to   = (Symbol *) line->to;
    Line        *copyline;

    /*
     *  copy line before deleting it, and restore the copy
     */

    copyline = (Line *) copy_line( line );

    /*
     *  set LDline[XY] to first cell in this line that is of line type
     */

    find_line_cell( tempseg );

    /*
     *  reclaim cells for each lineseg of line entering/leaving the bottom of
     *  symbol.  Delete line, then restore copy.
     */

    delete_line();

    tempseg = (LineSeg *) copyline->line;
    restore_line( tempseg, type, from->mycanvas, to->mycanvas );
    free_line( copyline );
}
```

```
/*************************************<----->*************************************
*
* MODULE NAME:  redraw_labels()
*
* MODULE FUNCTION:
*
*   This routine redraws the labels of an IF/SET symbol, since they may have been
*   obscured during the resizing.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*************************************<----->*************************************/

void  redraw_labels( symbol )

    Symbol  *symbol;

{
    XWindowAttributes   attribs;
    int                 redraw_label = 0,
                        x, y;

     if (! XGetWindowAttributes(display,XtWindow(symbol->mycanvas),&attribs) )
        error_handler( ERR_SYM_ATTRIBS, "redraw_labels" );

    /*
     *  if true or false label is obscured by new size,
     *  clear the old label and redraw it below symbol
     *  we can get away with checking only down because
     *  symbols currently only grow down
     */

    if ( symbol->Sym.IfSym.false_line )
        {
        if ( symbol->Sym.IfSym.false_line->line->orientation == DOWN )
            redraw_label = 1;
        }

     else if ( symbol->Sym.IfSym.true_line )
        {
        if ( symbol->Sym.IfSym.true_line->line->orientation == DOWN )
            redraw_label = 2;
        }

    /*
     *  label is within symbol - clear it and redraw below the
     *  lower edge
     */

    if ( redraw_label )
        {
        if ( redraw_label == 1 )
            clear_text( draw_area, symbol->Sym.IfSym.false_x,
                        symbol->Sym.IfSym.false_y,  "FALSE" );
        else
            clear_text( draw_area, symbol->Sym.IfSym.true_x,
                        symbol->Sym.IfSym.true_y, "TRUE" );

        x = symbol->ulcx + (attribs.width/2) + 10;
        y = symbol->ulcy + attribs.height + 10;

        if ( redraw_label == 1 )

        /*
         *  redrawing a false label.
         */

            {
            XDrawString( display, XtWindow(draw_area), LDgc, x, y, "FALSE", 5 );
            symbol->Sym.IfSym.false_x = x;
            symbol->Sym.IfSym.false_y = y;
            }
        else
            {
            XDrawString( display, XtWindow(draw_area), LDgc, x, y, "TRUE", 4 );
            symbol->Sym.IfSym.true_x = x;
            symbol->Sym.IfSym.true_y = y;
            }
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  redraw_last_lineseg()
*
* MODULE FUNCTION:
*
*   This routine checks all lines leaving out the bottom of an IF/SET symbol; redraws an
d
*   reclaims the ones that have been obscured during the resizing.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder ~ MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  redraw_last_lineseg( symbol )

    Symbol  *symbol;

{
    LineList    *templist, *nextlist;
    Symbol      *tempsymbol;
    LineSeg     *tempseg;

    /*
     *  If symbol has line leaving it out the bottom, reclaim its cells
     */

    if ( symbol->symbol_type == IF )
        {
        if ( symbol->Sym.IfSym.false_line )
                reclaim_line_cells( symbol->Sym.IfSym.false_line, 2 );

        if ( symbol->Sym.IfSym.true_line )
                reclaim_line_cells( symbol->Sym.IfSym.true_line, 1 );
        }
    else
        if ( (symbol->symbol_type==PRINT) || (symbol->symbol_type==SET) )
            if (symbol->next)
                reclaim_line_cells( symbol->next, 0 );
                }
    /*
     *  if symbol has line(s) entering it thru the bottom, reclaim its (their) cells
     */

    if ( symbol->from )
        {
        templist = (LineList *) symbol->from;
        nextlist = (LineList *) templist->next;
        while ( templist )
            {
            tempseg = templist->line->line;

            /*
             *  get to last segment in line; see if it is going up
             */

            tempsymbol = (Symbol*) templist->line->from;
            if ( tempsymbol->symbol_type == IF )
                {
                if (templist->line == (Line *) tempsymbol->Sym.IfSym.false_line)
                    reclaim_line_cells( templist->line, 2 );
                else if (templist->line == (Line *)tempsymbol->Sym.IfSym.true_line)
                    reclaim_line_cells( templist->line, 1 );
                }
            else if ( templist->line == (Line *) tempsymbol->next )
                reclaim_line_cells( templist->line, 0 );
            templist = (LineList *) nextlist;

            if ( nextlist )
                nextlist = (LineList *) nextlist->next;
            }
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  reinstall()
*
* MODULE FUNCTION:
*
*   This routine changes the size and content fields in the Symbol_Map entry of an
*   edited symbol.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  reinstall()
{
    XWindowAttributes   attribs;
    Symbol              *symbol;
    Arg                 args[2];
    int                 type,
                        attributes = 0;

    /*
     *  Try to claim cells of possibly expanded or shrunken symbol.
     */

    symbol = (Symbol *) get_sym_map_entry( current_symbol );

    XtSetArg( args[0], XmNuserData, &type );
    XtGetValues( current_symbol, args, 1 );

    if (! claim_cells(current_symbol) )
        {

        /*
         *  claim failed; reset links between text strings and symbols
         *  to symbol map contents, unchanged by unsuccessful claim attempt
         */

        if ( type == TEXT )
            user_ack("Sorry, can't put text there");
        else
            user_ack("Sorry, can't put symbol there");

        /*
         *  in case of if or set, reset comp_text
         */

        if ( (type == IF) || (type == SET) )
            {
            sym_text  = symbol->Sym.IfSym.logical_expr;
            expr_text = symbol->Sym.IfSym.comp_expr;
            }
        else
            sym_text = symbol->text;

        /*
         *  restore previous size; redraw symbol with old contents and size
         */

        XtSetArg( args[0], XmNwidth, symbol->width );
        XtSetArg( args[1], XmNheight, symbol->height );
        XtSetValues( current_symbol, args, 2 );

        draw_symbol( type, current_symbol, WAgc, symbol->font );
        return( 0 );
        }
    else
        {

        /*
         *  claim worked; set Symbol_Map fields.
         */

        symbol->font = WAFont;
        if ( (type == IF) || (type == SET) )
            {
            symbol->Sym.IfSym.logical_expr = sym_text;

            /*
             *  set comp and comment symbol map fields to current global values
             */

            symbol->Sym.IfSym.comp_expr = expr_text;
            symbol->Sym.IfSym.comment = comment_text;
            }

        if ( (type == START) || (type == STOP) )
            {
            if (! (struct symbol_entry *) lookup_symbol(NULL,sym_text) )
                {
                attributes |= COMP_REF;
                if ( add_symbol_entry(NULL,sym_text,attributes,0,0,0,0,0) )
                    error_handler( ERR_ADD_SYMBOL, "reinstall" );
                }

            /*DEBUG*/
            elog(3,"reinstall: for comp name %s, incr use count to %i",
            sym_text, increment_symbol_use_count( NULL, sym_text ) );

            elog(3,"reinstall: for comp name %s, decr use count to %i",
            symbol->text, decrement_symbol_use_count( NULL, symbol->text ) );
            }

        if ( type == GOTO )
            {
            symbol->Sym.ElemSym.comp_type = CallType;
            if (! (struct symbol_entry *) lookup_symbol(NULL, sym_text) )
                {
                /*DEBUG*/
               elog(3,"reinstall: element name %s not found; adding to sym tab",
                    sym_text);

                attributes |= PROCEDURE;
                if ( add_symbol_entry(NULL,sym_text,attributes,0,0,0,0,0) )
                    error_handler( ERR_ADD_SYMBOL, "reinstall" );
                }

            /*DEBUG*/
            elog(3,"reinstall: for element name %s, incr use count to %i",
            sym_text, increment_symbol_use_count( NULL, sym_text ) );

            elog(3,"reinstall: for element name %s, decr use count to %i",
            symbol->text, decrement_symbol_use_count( NULL, symbol->text ) );
```

```
        }

    symbol->text = sym_text;

    /*
     *  redraw lines entering or leaving symbol bottom that symbol
     *  has grown over.
     */

    if ( (type == IF) || (type == SET) || (type == PRINT) )
        {

        /*DEBUG*/
        elog(3,"reinstall: checking lines");

        check_lines( symbol );
        redraw_last_lineseg( symbol );
        }

    if ( type == IF )
        redraw_labels( symbol );

    return( 1 );
    }
}
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    cbr_help.c
*
* FILE FUNCTION:
*
*   This file contains the routines which build the Help Text popup and the routines
*   which get and display the help text when the user clicks HELP.
*
*
* FILE MODULES:
*
*   build_help()     - build the Help Text viewer popop
*   cbr_help()       - pops up help window. Contents depend on selected menu item
*
*****************************************<----->*****************************************/

#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>

#include "qcb.h"
#include "cbr.h"
#include "widgets.h"
#include "constants.h"
#include "pixmaps.h"
#include "tokens.h"

XmString    pal_header,
            blank_header;

char    help_help_str[] =

"This popup allows the user to view the help text \nfor the Graphical Comp Builder.  The
 user may use the scroll \nbars to scroll through the help text.  Select close \nto clos
e this window and return to the help text.";
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  build_help()
*
* MODULE FUNCTION:
*
*   This routine builds the popup which is used to display Help text to the user.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  build_help( parent )

{
    Pixmap  pixmap;
    Pixel   background,
            foreground;
    XImage  *image;
    Arg     args[2];

    /*
     *  create context_dependent headers for popup.
     */

    pal_header   = XmStringCreate( "Click on palette items for more help",
                                    XmSTRING_DEFAULT_CHARSET );
    blank_header = XmStringCreate( " ", XmSTRING_DEFAULT_CHARSET );

    /*
     *  create display file popup
     */

    dlg_file  = cr_popup( NULLS, parent, "Help" );
    FormW     = cr_form ( NULLS, dlg_file, NULL, NULL );
    txt_file  = cr_text ( NULLS, FormW, NULL, NULL,"", TRUE, 20, 65 );

    /*
     *   set help text widget ineditable.
     */

    XtSetArg( args[0], XmNeditable, FALSE );
    XtSetValues( txt_file, args, 1 );

    XtSetArg( args[0], XmNforeground, &foreground );
    XtSetArg( args[1], XmNbackground, &background );
    XtGetValues( dlg_file, args, 2 );

    image = (XImage *) CreateDefaultImage( infoBits, 11, 24 );
    XmInstallImage( image, "info_img" );
    pixmap = XmGetPixmap( XtScreen(dlg_file), "info_img", foreground, background );

    pix_help  = cr_pixmap( NULLS, FormW, &pixmap, 75, IGNORE, 10, IGNORE ) ;
    lbl_file  = cr_label ( NULLS, FormW, " ", 0, 76, 80, 25, IGNORE );

    cr_separator( NULLS, FormW, 87, 89, 0, 100 );

    Ncr_rel_cmd ( NULLS, FormW, "Close", cbr_cancel, FILE_OK,   93, 10 );
    Ncr_rel_cmd ( NULLS, FormW, "Help",  cbr_help,   HELP_HELP, 93, 70 );
```

```
    /*
     *  create display help for help popup
     */

    dlg_help = cr_popup( NULLS, dlg_file, "Help for Help" );
    FormW    = cr_form ( NULLS, dlg_help, NULL, NULL );

    cr_label( NULLS, FormW, help_help_str, 0, 10, IGNORE, 5, 95 );
    cr_separator( NULLS, FormW, 65, 70, 0, 100 );

    Ncr_rel_cmd( NULLS, FormW,"Close", cbr_cancel, HELP_HELP_CANCEL, 81, 38 );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_help()
*
* MODULE FUNCTION:
*
*   This routine pops up a help window. The contents of the window depend on which
*   menu item in the help menu the user has selected.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_help( w, help_index, call_data )

    Widget  w;
    int     help_index;
    caddr_t call_data;

{
    Arg     args[5];

    /*
     *  The user wants HELP on the HELP text displayer.
     */

    if ( help_index == HELP_HELP )
        {
        XtManageChild( dlg_help );
        return;
        }

    /*
     *  If the user has selected the help palette area button, change the mode so
     *  that when one of the palette items is clicked, text specific to that item
     *  appears in a popup.
     */

    if ( help_index == PALETTE_AREA )
        {
        XtManageChild( pix_help );
        Mode  = Help;
        upd_mode_panel();
        XtSetArg( args[0], XmNdialogStyle, XmDIALOG_MODELESS );
        XtSetValues( dlg_file, args, 1 );
        XtSetArg( args[0], XmNlabelString, pal_header );
        }
    else
        {
        XtUnmanageChild( pix_help );
        XtSetArg( args[0], XmNlabelString, blank_header );
        }
    XtSetValues( lbl_file, args, 1 );

    /*DEBUG*/
    elog(3,"cbr help: cd = %i", help_index);

    /*
     *  build a temporary file containing help text for the selected subject,
```

```
     *  and display it in the popup. Load_help_file is passed a string to search
     *  for in the help file.
     */

    load_help_file( tokens[help_index] );
    display_file( HELP, "/tmp/gcb.tmp" );

    /*
     *  Delete the temporary file.
     */

    if ( system("rm /tmp/gcb.tmp >>/tmp/gcb.err 2>&1") != OK )
        {
        user_ack("Unable to delete: /tmp/code.tmp - help text temporary file");
        elog(1,"Unable to delete: /tmp/code.tmp - help text temporary file");
        }
}
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    cbr_if.c
*
* FILE FUNCTION:
*
*   This file contains the routines which  pop up if/set symbol creation popups.
*
*
* FILE MODULES:
*
*   build_log_attribs( ) - builds popup to input font and size of logical symbols.
*   build_log_popup()    - builds Logical Expression popup, to enter logic expressions
*   cbr_attribs_done()   - takes down the attributes popup.
*   cbr_parse()          - responds when user presses the "parse expression" button.
*   cbr_if()             - responds when user presses the IF or SET button.
*
*****************************************<----->*****************************************/

#include <stdio.h>
#include <X11/Intrinsic.h>

#include <Xm/Xm.h>
#include <Xm/MwmUtil.h>
#include <Xm/SelectioB.h>

#include "gcb.h"
#include "cbr.h"
#include "widgets.h"
#include "symbol.h"
#include "gcb_parse.h"
#include "next_inputs.h"
#include "constants.h"
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  build_log_attribs()
*
* MODULE FUNCTION:
*
*   This routine builds the popup to input the font and symbol size of logical
*   symbols.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                                  Release 1.01 - 08/01/91
*                                  Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  build_log_attribs( parent )

    Widget parent;

{
    dlg_logic_attribs = cr_popup( NULLS, parent, "Symbol Attributes Input" );

    FormW = cr_form( NULLS, dlg_logic_attribs, NULL, NULL );
    set_attribs( FORM, FormW, 300, 250, XmRESIZE_NONE );

    cr_label( NULLS, FormW, "Symbol Size:", 0, 31, IGNORE, 15, IGNORE);

    rb_sym_size = cr_radio_box( NULLS, FormW, XmHORIZONTAL );

    tgl_sym_size_lg = cr_toggle( NULLS, rb_sym_size, "Large", NULL, 0, 0);
    tgl_sym_size_sm = cr_toggle( NULLS, rb_sym_size, "Small", NULL, 0, 0);
    arm_tgl( tgl_sym_size_sm );

    cr_label( NULLS, FormW, "Font Size:", 0, 45, IGNORE, 15, IGNORE);

    rb_font_size = cr_radio_box( NULLS, FormW, XmHORIZONTAL );

    tgl_font_size_lg = cr_toggle( NULLS, rb_font_size, "Large", NULL, 1, 1 );
    tgl_font_size_sm = cr_toggle( NULLS, rb_font_size, "Small", NULL, 2, 2 );
    arm_tgl( tgl_font_size_sm );

    cr_separator( NULLS, FormW, 80, IGNORE, 1, 99 );

    CancelW = cr_command( NULLS, FormW, "Cancel", cbr_attribs_done, 0 );
    DoneW   = cr_command( NULLS, FormW, "Done",   cbr_attribs_done, 1 );
    HelpW   = cr_command( NULLS, FormW, "Help",   cbr_help, SYM_ATTR_HELP );

    set_position( rb_sym_size,  28, IGNORE, 50, IGNORE);
    set_position( rb_font_size, 42, IGNORE, 50, IGNORE);
    set_position( CancelW,      90, IGNORE,  2, IGNORE);
    set_position( DoneW,        90, IGNORE, 35, IGNORE);
    set_position( HelpW,        90, IGNORE, 70, IGNORE);
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  build_log_popup()
*
* MODULE FUNCTION:
*
*   This routine builds the Logical Expression popup.  The popup built by this routine
*   is used to enter the logical expressions in DECISION and SET symbols.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.01 - 08/01/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  build_log_popup( parent )

    Widget parent;

{
    Arg     args[5];

    dlg_logic = cr_popup( NULLS, parent, "Logical Expression Input" );
    XtSetArg( args[0], XmNdialogStyle, XmDIALOG_MODELESS );
    XtSetValues( dlg_logic, args, 1 );

    FormW = cr_form(NULLS, dlg_logic, NULL, NULL );
    set_attribs( FORM, FormW, 500, 650, XmRESIZE_NONE );

    scr_logic = cr_scr_text( NULLS, FormW, 8, True, 250, 175, 25 );
    XtSetArg( args[0], XmNscrollingPolicy, XmAUTOMATIC );
    XtSetValues( scr_logic, args, 1 );

    scr_expr = cr_scr_text( NULLS, FormW, 8, True, 250, 175, 200 );
    XtSetArg( args[0], XmNscrollingPolicy, XmAUTOMATIC );
    XtSetValues( scr_expr, args, 1 );

    scr_comment = cr_scr_text( NULLS, FormW, 10, True, 450, 30, 400 );
    XtSetArg( args[0], XmNscrollingPolicy, XmAUTOMATIC );
    XtSetValues( scr_comment, args, 1 );

    /*
     *  create labels for text windows
     */

    cr_label( NULLS, FormW, "Logic Description:",    0, 6,IGNORE,6,IGNORE );
    cr_label( NULLS, FormW, "Comp Expression:",      0,33,IGNORE,6,IGNORE );
    cr_label( NULLS, FormW, "Description:",          0,56,IGNORE,6,IGNORE );

    /*
     *  create button for parsing expr
     */

    btn_logic_parse = cr_command(NULLS, FormW, "Parse Expression", cbr_parse, 0);

    /*
     *  create buttons for done and cancel
     */

    cr_separator( NULLS, FormW, 91, IGNORE, 1, 99 );

    CancelW           = cr_command( NULLS, FormW, "Cancel",     cbr_cancel,2);
    btn_logic_attribs = cr_command( NULLS, FormW, "Attributes", cbr_done,  0);
    DoneW             = cr_command( NULLS, FormW, "Done",       cbr_done,  1);
    HelpW             = cr_command( NULLS, FormW, "Help",       cbr_help,  LOGIC_POPUP);

    set_position( btn_logic_parse,  38, IGNORE,  6, IGNORE);
    set_position( CancelW,          96, IGNORE,  5, IGNORE );
    set_position( btn_logic_attribs,96, IGNORE, 29, IGNORE );
    set_position( DoneW,            96, IGNORE, 54, IGNORE );
    set_position( HelpW,            96, IGNORE, 80, IGNORE );

    /*
     *  Build the logical expression sub-popups...
     */

    /*
     *  Build the variable declaration sub-popup.
     */

    build_local_input       ( dlg_logic );

    /*
     *  Build the number input sub-popup.
     */

    build_var_input_popup   ( dlg_logic );

    /*
     *  Build the defined function selection sub-popup.
     */

    build_def_fn_input_popup( dlg_logic );

    /*
     *  Build the string sub-popup.
     */

    build_str_popup         ( dlg_logic );

    /*
     *  Build the sub-popup to input types of undeclared variables
     */

    build_unknown_type_popup( dlg_logic );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_attribs_done()
*
* MODULE FUNCTION:
*
*   This routine takes down the attributes popup.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_attribs_done( w, client_data, unused)

    Widget                  w;
    int                     client_data;
    caddr_t                 unused;

{
    XtUnmanageChild( dlg_logic_attribs );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_parse()
*
* MODULE FUNCTION:
*
*   This routine responds when user presses the "parse expression" button.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_parse( w, client_data, unused )

    Widget                  w;
    int                     client_data;
    caddr_t                 unused;

{
    Arg     args[1];
    int     type, i;
    char    response[80];

    if ( !strcmp(XmTextGetString(scr_expr), NULL) )
        {

        /*
         *  no text to parse, initialize buttons according to symbol type.
         */

        XtSetArg( args[0], XmNuserData, &type );
        XtGetValues( current_symbol, args, 1 );
        if ( type == IF )
            init_inputs( -1 );
        else init_inputs( -2 );
        invalidate_buttons();
        return;
        }

    /*
     *  determine types of all vars, then verify types in expression.
     */

    if ( build_new_var_list(XmTextGetString(scr_expr), (Mode==AddSymbol) ? 1:0) )
        {
        if ( (i = verify_expression_type(XmTextGetString(scr_expr), response)) !=
            VALID_EXPRESSION )
                {
                user_ack( response );
                return;
                }
        }
    else return;

    do_parse();
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_if()
*
* MODULE FUNCTION:
*
*   This routine handles events when the user presses the IF or SET button.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_if( w, client_data, unused )

    Widget                  w;
    caddr_t                 client_data;
    caddr_t                 unused;

{
    Arg                     args[2];
    XWindowAttributes       wattribs;
    XmString                eq_string;

    /*
     *  if mode is already add symbol, cancel the add and reset the mode.
     */

    if ( Mode == AddSymbol )
        {
        cancel_draw();
        return;
        }

    /*
     *  if we are in help palette area mode, popup help for the chosen
     *  palette item.
     */

    if ( Mode == Help )
        {
        cbr_help( w, (int)client_data, unused );
        return;
        }

    if ( Mode != EditSymbol )
        return;

    /*
     *  Make sure we have a valid element, can't add a symbol if we don't
     *  have a valid element.
     */

    if (! ValidElement )
        {
        error_handler( NO_ELEMENT, NULL );
        return;
        }

    /*
     *  Currently, the GCB does not allow the user to add symbols which affect
     *  the Comp symbol table to be added to a Library Element.
     */

    if ( ElementType == LIB )
        {
        error_handler( LIB_ELEM_SYM, NULL );
        return;
        }

    /*
     *  record last chosen symbol type.
     */

    invert( (int)client_data );

    /*
     *  See if we have data structures for another symbol.
     */

    if ( (next_avail_sym()) < 0 )
        return ;

    Mode = AddSymbol;
    upd_mode_panel();

    /*
     *  Set the dimensions of the symbol according to its type and whether or
     *  not we are zoomed.
     */

    if (! XGetWindowAttributes(display,XtWindow(w),&wattribs) )
        error_handler( ERR_SYM_ATTRIBS, "cbr_if" );

    XtSetArg( args[0], XmNwidth, (Zoomed)?op(wattribs.width+80, 2) :
        wattribs.width + 80 );
    XtSetArg( args[1], XmNuserData, (int)client_data );
    XtSetValues( current_symbol, args, 2);

    /*
     *  initialize expression input buttons according to type of
     *  expression being created.
     */

    if ( (int)client_data == IF )
        init_inputs( -1 );
    else
        init_inputs( -2 );
    invalidate_buttons();

    /*
     *  Set "equals" button to '=' or ':=' according to whether we are
     *  in a SET or an IF.
     */

    XtSetArg(args[0], XmNlabelString, &eq_string);
    XtGetValues(eq_btn, args, 1);
    XmStringFree(eq_string);

    if ( (int)client_data == IF )
        eq_string = XmStringCreate( "=",    XmSTRING_DEFAULT_CHARSET );
    else
        eq_string = XmStringCreate( ":=",    XmSTRING_DEFAULT_CHARSET );
```

```
    XtSetArg( args[0], XmNlabelString, eq_string );
    XtSetValues( eq_btn, args, 1);
    XmStringFree( eq_string );

    /*
     *  initialize global variables so parser knows where it is and what
     *  type of expression it is parsing.
     */

    WhereAmI = LHS;
    paren_count = 0;
    if ( (int) client_data == SET )
        SetSym = 1;
    else
        SetSym = 0;

    /*
     *  take down the palette items window and overlay its space  with
     *  the expression input buttons.
     */

    XtUnmapWidget( frame_palette );
    XtMapWidget( frame_math_menu );
    XtManageChild( dlg_logic );
}
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    cbr_menu.c
*
* FILE FUNCTION:
*
*   This file contains the routines which respond to events in the popup and
*   pulldown menus.
*
*
* FILE MODULES:
*
*   alter_endpts()        - increases or decreases endpts of the segments of a line
*   alter_final_endpts()- alters final endpt of line so that it is not in a symbol.
*   cbr_set_anchor()      - sets the mode of the box being drawn and changes cursor.
*   check_zoomed_line()  - ensures that zoomed in/out line doesnt encroach on anything
*   cvt_to_str_array()    - surrounds each word in the parameter string with parens
*   delete_box()          - deletes all symbols and lines in banded box.
*   menu_proc()           - responds to events in the popup menu.
*   op()                  - sets zoom units.
*   pix_centering_height()  - sets the margins for postscript output.
*   pix_right_margin()    - sets the margins for postscript output.
*   pix_left_margin()     - sets the margins for postscript output.
*   pix_to_ps()           - converts postscript units to pixels and vice versa.
*   print_arrows()        - prints line arrows in postscript.
*   print_from_lines()    - prints in postscript the lines that enter each symbol.
*   print_symbols()       - prints in postscript the form and text of each symbol.
*   set_arrows()          - sets the proper endpts of a line to show the arrows.
*   zoom()                - zooms in or out.
*   zoom_symbols()        - resizes all the symbols, then redraws them.
*   zoom_lines()          - resizes all the lines, then redraws them.
*
*****************************************<----->*****************************************/

#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>

#include "gcb.h"
#include "cbr.h"
#include "widgets.h"
#include "menu.h"
#include "lines.h"
#include "constants.h"
#include "fonts.h"
#include "cursors.h"
#include "print.h"

#define IN  1
#define OUT 2
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  alter_endpts()
*
* MODULE FUNCTION:
*
*   This routine increases or decreases the endpts of the segments of a line
*   during a zoom.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  alter_endpts( line, size )

    Line        *line;
    int         size;

{
    LineSeg     *tempseg = (LineSeg *)line->line;
    void        alter_final_endpt();

    /*
     *  Get to beginning of list
     */

    while ( tempseg->prev )
        tempseg = (LineSeg *)tempseg->prev;

    /*
     *  Alter all the endpts of each segment
     */

    while ( tempseg )
        {
        tempseg->end_x = op( tempseg->end_x, size );
        tempseg->end_y = op( tempseg->end_y, size );
        tempseg->start_y = op( tempseg->start_y, size );
        tempseg->start_x = op( tempseg->start_x, size );

        tempseg->cell_end_x = tempseg->end_x / CELL_SIZE;
        tempseg->cell_end_y = tempseg->end_y / CELL_SIZE;
        tempseg->cell_start_x = tempseg->start_x / CELL_SIZE;
        tempseg->cell_start_y = tempseg->start_y / CELL_SIZE;

        if ( !(tempseg->next) )

            /*
             *  last lineseg of line. alter endpts so that the
             *  lineseg's last pt isnt within the symbol it connects.
             */

            alter_final_endpt( tempseg, line );

        tempseg = (LineSeg *)tempseg->next;
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  alter_final_endpt()
*
* MODULE FUNCTION:
*
*   This routine alters the final endpt of a line so that it is not within
*   the destination symbol.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  alter_final_endpt( lineseg, line )

    LineSeg     *lineseg;
    Line        *line;

{
    Symbol      *to = (Symbol *)line->to;

    /*
     *  last lineseg of line. alter endpts so that the
     *  lineseg's last pt isnt within the symbol it connects.
     */

    switch( lineseg->orientation )
        {

        /*
         *  depending on orientation, set final cell_[xy]
         */

        case UP:
            lineseg->cell_end_y = to->cell_y + to->cell_height + 1;
            break;
        case DOWN:
            lineseg->cell_end_y = to->cell_y - 1;
            break;
        case LEFT:
            lineseg->cell_end_x = to->cell_x + to->cell_width + 1;
            break;
        case RIGHT:
            lineseg->cell_end_x = to->cell_x - 1;
            break;
        }
}
```

```
/*************************************<----->*************************************
*
* MODULE NAME:  cbr_set_anchor()
*
* MODULE FUNCTION:
*
*   This routine sets the mode of the box being drawn and changes the cursor.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*************************************<----->*************************************/

void  cbr_set_anchor( w, client_data, call_data )

    Widget  w;
    int     client_data;
    caddr_t call_data;

{
    /*
     *  user wants to draw a print, delete, copy, or move box. Set the mode and
     *  change the cursor.
     */

    if ( Mode == EditSymbol )
        {
        if ( client_data == PrintBox )
            Mode = PrintBox;
        else if ( client_data == DeleteBox )
            Mode = DeleteBox;
        else if ( client_data == CopyBox )
            Mode = CopyBox;
        else if ( client_data == MoveBox )
            Mode = MoveBox;
        upd_mode_panel();
        XDefineCursor( display, XtWindow(draw_area), ul_cursor );
        }
}
```

```
/*************************************<----->*************************************
*
* MODULE NAME:  change_line()
*
* MODULE FUNCTION:
*
*   This routine changes the endpts of each seg of a line, then redraws it with
*   new endpoints.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*************************************<----->*************************************/

void  change_line( lineptr, size )

    Line    *lineptr;
    int     size;

{
    void        alter_endpts();
    int         line_type, x, y;
    LineSeg     *temp_line = lineptr->line, *labelPtr;
    Symbol      *from = (Symbol *)lineptr->from;
    void        set_arrows();

    /*
     *  change endpts of linesegs of lines entering symbol; then redraw
     *  them.
     */

    if ( lineptr != NULL )
        {

        /*
         *  alter endpts of each lineseg
         */

        alter_endpts( lineptr, size );

        if ( from->symbol_type == IF )
            {

            /*
             *  determine which kind of line we are drawing.
             */

             if ( from->Sym.IfSym.true_line == lineptr )
                line_type = 1;
             else if ( from->Sym.IfSym.false_line == lineptr )
                line_type = 2;
            else
                {
                user_ack("line doesn't match either true or false line of from");
                elog(1,"line doesn't match either true or false line of from");
                exit( ERR );
                }
            }
        else
            line_type = 0;
```

```
        /*
         *  Go thru linesegs; for each, Set LD[start stop][XY] and redraw line.
         */

        while ( temp_line )
            {
            LDstartX = temp_line->start_x;
            LDstartY = temp_line->start_y;
            LDendX   = temp_line->end_x;
            LDendY   = temp_line->end_y;
            set_line();
            if ( !(temp_line->next) )
                {

                /*
                 *  final segment; set arrow pt and draw arrows
                 */

                set_arrows( temp_line );
                if ( line_type )
                    {
                    if ( line_type == 1 )
                        labelPtr = (LineSeg *)from->Sym.IfSym.true_line->line;
                    else if ( line_type == 2 )
                        labelPtr = (LineSeg *)from->Sym.IfSym.false_line->line;
                    else
                        {
                        user_ack("invalid line type in change_line!");
                        exit( ERR );
                        }

                     /*
                      *  Determine where to put the label.
                      */

                    if ( labelPtr->end_x > labelPtr->start_x )
                        {

                        /*
                         *  horizontal line moving to the right.
                         */

                        x = from->ulcx + from->width  + 2;
                        y = labelPtr->start_y - 10;
                        }
                    else if ( labelPtr->end_x < labelPtr->start_x )
                        {

                        /*
                         *  horizontal line moving to the left.
                         */

                        x = from->ulcx - 40;
                        y = labelPtr->start_y - 10;
                        }
                    else if ( labelPtr->end_y > labelPtr->start_y )
                        {
                        x = labelPtr->start_x + 10;
                        y = from->ulcy + from->height + 10;
                        }
                    else if ( labelPtr->end_y < labelPtr->start_y )
                        {
                        x = labelPtr->start_x + 10;
                        y = from->ulcy - 5;
                        }

                    /*
                     *  Draw label.
                     */

                    if ( line_type == 1 )
                        XDrawString( display, XtWindow(draw_area), LDgc, x, y,
                            "TRUE",strlen("TRUE") );
                    else if ( line_type == 2 )
                        XDrawString( display, XtWindow(draw_area), LDgc,
                            x,y,"FALSE",strlen("FALSE") );
                    }

                     /*
                      *  Record label in symbol structure.
                      */

                    if ( line_type == 1 )
                        {
                        from->Sym.IfSym.true_x = x;
                        from->Sym.IfSym.true_y = y;
                        }
                    else
                        {
                        from->Sym.IfSym.false_x = x;
                        from->Sym.IfSym.false_y = y;
                        }
                    }
                temp_line = (LineSeg *)temp_line->next;
                }
            }
        }
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  check_zoomed_line()
*
* MODULE FUNCTION:
*
*   This routine ensures that a zoomed in/out line doesnt encroach on anything else.
*   This can happen because the relative size of symbols may change in zoomed mode
*   due to Motif's inability to scale fonts.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  check_zoomed_line( line )

    Line    *line;

{
    LineSeg     *tempseg;
    int         i, line_length;

    /*
     *  for each segment in the line, check each cell to see if it is already
     *  occupied. If so, return error.
     */

    tempseg = (LineSeg *)line->line;
    while (tempseg)
        {

        /*
         *  check only horizontal line segments; vertical segments can't be
         *  grown over by symbols.
         */

        if ( (tempseg->orientation != UP) &&
            (tempseg->orientation != DOWN) )
            {
            line_length = abs(tempseg->cell_end_x - tempseg->cell_start_x);
            for (i=tempseg->cell_start_x; i<line_length;
                (tempseg->orientation == RIGHT) ? i++ : i--)
                {
                if (Cell_Map[tempseg->cell_start_y][i].cell_type == SYMBOL_CELL)
                    if ( (Cell_Map[tempseg->cell_start_y][i].cell_entry.symbol
                        !=  (Symbol *)line->from)
                            &&
                        (Cell_Map[tempseg->cell_start_y][i].cell_entry.symbol
                            != (Symbol *)line->to) )
                                return( 0 );
                }
            }
            tempseg = (LineSeg *)tempseg->next;
        }
    return( 1 );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cvt_to_str_array()
*
* MODULE FUNCTION:
*
*   This routine  surrounds each word in the parameter string with parens --
*   makes postscript creation easier.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

char *cvt_to_str_array(string, result)

    char *string, *result;

{
    char *p = string;
    int i = 1;

    /*
     *  if string is null, return (( ))
     */

    if ( string == NULL )
        {
        result[0] = '[';
        result[1] = '(';
        result[2] = ' ';
        result[3] = ')';
        result[4] = ']';
        result[5] = '\0';
        return;
        }

    result[0] = '[';
    while (*p)
        {
        result[i] = '(';

        /*
         *  parens themselves are special cases -- make them separate strings
         */

        if ( (*p == ')') || (*p == '(') )
            {
                result[++i] = '\\';
                result[++i] = *p;
            }

        /*
         *  put in marker for newline
         */

        else if (*p == '\n')
            {
            result[++i] = '@';
            result[++i] = '@';
```

```
        }

    else if (*p == ' ')
        {
        result[++i] = ' ';
        }

    else

    /*
     *  word - skip until end of word.
     */

        {
        while ( (*p) && (*p != '\n') && (*p != ' ')
            && (*p != ')')  && (*p != '(') )
            {
                result[++i] = *p++;
            }
        *p--;
        }

    /*
     *  terminate in paren
     */

    result[++i] = ')';

    /*
     *  advance pointer along string
     */

    if (*p)
        {
        *p++;
        i++;
        }
    }
result[i++] = '}';
result[i] = '\0';
return( result );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  delete_box()
*
* MODULE FUNCTION:
*
*   This routine  deletes all symbols and lines in banded box.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  delete_box( b_ulcx, b_ulcy, b_lrcx, b_lrcy, prompt)

    int b_ulcx, b_ulcy, b_lrcx, b_lrcy, prompt;
{
    int i, lrcx, lrcy, to_be_deleted = 0;

    /*
     *  give the user a chance to recant.
     */

    if ( (!prompt) || (ask("No Undo: OK to delete?")) )
        {

        /*
         *  go through Symbol_Map; if a symbol's dimensions are
         *  at least partially within the delete box, delete the
         *  symbol.
         */

        for (i=0; i<MAX_SYMBOLS; i++)
            {
            if (Symbol_Map[i].symbol_type != NONE)
                {

                /*
                 *  determine lower right corner of symbol
                 */

                lrcx = Symbol_Map[i].ulcx + Symbol_Map[i].width;
                lrcy = Symbol_Map[i].ulcy + Symbol_Map[i].height;

                /*
                 *  if either left or right coordinate of the symbol within box ...
                 */

                if ( ( (lrcx > b_ulcx) && (lrcx < b_lrcx) ) ||
                  ( (Symbol_Map[i].ulcx > b_ulcx)&&(Symbol_Map[i].ulcx < b_lrcx) ))
                    {

                    /*
                     *   if either top or bottom coordinate of the symbol within
                     *   box, delete symbol.
                     */

                    if ( (lrcy > b_ulcy) && (lrcy < b_lrcy) )
                        to_be_deleted = TRUE;
                    else if ((Symbol_Map[i].ulcy > b_ulcy) &&
```

```
                    (Symbol_Map[i].ulcy<b_lrcy))
                        to_be_deleted = TRUE;
                    }
                if ( to_be_deleted )
                    remove_symbol( Symbol_Map[i].mycanvas );
                to_be_deleted = FALSE;
                }
            }       /*for*/
        }           /*if*/
    else

        /*
         *  user has recanted; just redraw the symbols in proper colors.
         */

        clear_audit();

    Mode = EditSymbol;
    upd_mode_panel();
}
```

```
/*****************************************<----->*************************************
*
* MODULE NAME:  highlight_box()
*
* MODULE FUNCTION:
*
*   This routine highlights all symbols and lines wholely within the parameter box.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*************************************/

void  highlight_box( b_ulcx, b_ulcy, b_lrcx, b_lrcy)

    int b_ulcx, b_ulcy, b_lrcx, b_lrcy;
{
    int i, lrcx, lrcy;
    int to_be_highlighted = 0;

    /*
     *  go through Symbol_Map; if a symbol's dimensions are
     *  at least partially within the delete box, highlight the
     *  symbol and its lines.
     */

    for (i=0; i<MAX_SYMBOLS; i++)
        {
        if (Symbol_Map[i].symbol_type != NONE)
            {

            /*
             *  determine lower right corner of symbol
             */

            lrcx = Symbol_Map[i].ulcx + Symbol_Map[i].width;
            lrcy = Symbol_Map[i].ulcy + Symbol_Map[i].height;

            /*
             *  if either left or right coordinate of the symbol within box ...
             */

            if ( ( (lrcx > b_ulcx) && (lrcx < b_lrcx) ) ||
              ( (Symbol_Map[i].ulcx > b_ulcx)&&(Symbol_Map[i].ulcx < b_lrcx) ))
                {

                 /*
                  *    if either top or bottom coordinate of the symbol within box,
                  *    highlight symbol.
                  */

                 if ( (lrcy > b_ulcy) && (lrcy < b_lrcy) )
                      to_be_highlighted = TRUE;
                 else if ((Symbol_Map[i].ulcy > b_ulcy)&&(Symbol_Map[i].ulcy<b_lrcy))
                      to_be_highlighted = TRUE;
                 }
            if (to_be_highlighted)
                {
                highlight_sym( &Symbol_Map[i]);
```

```
            highlight_sym_lines( &Symbol_Map[i], FALSE);
            }
        to_be_highlighted = FALSE;
        }
    }       /*for*/
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  menu_proc()
*
* MODULE FUNCTION:
*
*   This routine responds to events in the popup menu.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  menu_proc(w, closure, call_data)

    Widget  w;
    int     closure;
    caddr_t call_data;

{
    void    zoom();

    if ( closure == 3 )
        {

        /*
         *  zoom only in editsymbol mode
         */

        if ( Mode == EditSymbol )
            {
            if ( Zoomed )
                zoom( IN );
            else
                zoom( OUT );
            }
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  op()
*
* MODULE FUNCTION:
*
*   This routine sets zoom units.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int op(point, size)

    int point, size;

{
    /*
     *  if we are zooming out, return a fraction of the unit
     */

    if ( size == OUT )
        return( point * 8 / 12 );
    else

    /*
     *  if we are zooming in, return a multiple of the unit
     */

        return( point * 12 / 8 );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  pix_centering_height()
*               pix_right_margin()
*               pix_left_margin()
*
* MODULE FUNCTION:
*
*   These routines set the margins for postscript output.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  pix_centering_height()

{
    /*
     *  postscript vertical units start from the bottom of the page, X vertical
     *  units start from the top; this routine compensates for postscript by
     *  returning the offset necessary to subtract from X units.
     */

    if ( PrintScale == REDUCED )
        return( 1300 );
    else
        return( 830 );
}


int  pix_right_margin()

{
    if ( PrintScale == REDUCED )
        return( 1500 );
    else
        return( 689 );
}


int  pix_left_margin()

{
    if ( PrintScale == REDUCED )
        return( 200 );
    else
        return( 80 );
}
```

```
/*****************************************<-----*****************************************
*
* MODULE NAME:  pix_to_ps()
*
* MODULE FUNCTION:
*
*   This routine  converts postscript units to pixels and vice versa.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<-----*****************************************/

int  pix_to_ps(pixels)

    int pixels;

{
    /*
     *  translate pixel coordinates into ps units
     *  position in inches * 72 is position in PostScript units
     */

    pixels = pixels * 72;
    if ( PrintScale == REDUCED )
        return( (int) pixels / 180.0 );
    else
        return( (int) pixels / 90.0 );
}
```

```
/*****************************************<-----*****************************************
*
* MODULE NAME:  print_arrows()
*
* MODULE FUNCTION:
*
*   This routine  prints line arrows in postscript.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<-----*****************************************/

void  print_arrows( endx, endy, orientation, fp )

    int endx, endy, orientation;
    FILE    *fp;
{
    /*
     *  depending on orientation, print arrow at endpt
     */

    fprintf( fp, "%d %d moveto\n", endx, endy);
    if ( orientation == RIGHT )
        {
        fprintf( fp, "%d %d lineto\n", endx - 6, endy + 6 );
        fprintf( fp, "%d %d moveto\n", endx, endy );
        fprintf( fp, "%d %d lineto\n", endx - 6, endy - 6 );
        }
    else if ( orientation == LEFT )
        {
        fprintf( fp, "%d %d lineto\n", endx + 6, endy + 6 );
        fprintf( fp, "%d %d moveto\n",  endx, endy );
        fprintf( fp, "%d %d lineto\n", endx + 6, endy - 6 );
        }
    else if ( orientation == UP )
        {
        fprintf( fp, "%d %d lineto\n",  endx + 6, endy - 6 );
        fprintf( fp, "%d %d moveto\n",  endx, endy );
        fprintf( fp, "%d %d lineto\n",  endx - 6, endy - 6 );
        }
    else if ( orientation == DOWN )
        {
        fprintf( fp, "%d %d lineto\n", endx + 6, endy + 6 );
        fprintf( fp, "%d %d moveto\n", endx, endy );
        fprintf( fp, "%d %d lineto\n", endx - 6, endy + 6 );
        }
    fprintf( fp, "stroke\n" );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  print_from_lines()
*
* MODULE FUNCTION:
*
*   This routine  prints in postscript the lines that enter each symbol.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  print_from_lines( linelist, fp, width, height, xoffset, yoffset )

    LineList    *linelist;
    FILE        *fp;
    int         width, height, xoffset, yoffset;

{
    LineSeg     *lineseg;
    Line        *line;
    Symbol      *dest;
    int         from_ifsymbol = 0, labelx, labely, startx, starty, endx, endy;

    /*
     *  go thru each symbol's 'from' line list.
     *  for each line, set the start point for the first segment, since
     *  it currently is in the middle of the symbol.
     */

    while ( linelist )
        {
        line = (Line *)linelist->line;
        lineseg = (LineSeg *)line->line;

        dest = (Symbol *)line->from;
        if ( dest->symbol_type == IF )
            from_ifsymbol = 1;

        switch ( lineseg->orientation )
            {
            case RIGHT:

                /*
                 *  start drawing the line at the rightmost edge of the symbol
                 */

                startx = pix_to_ps( lineseg->start_x + dest->width/2 - xoffset );
                starty = pix_to_ps( pix_centering_height() - (lineseg->start_y -
                    yoffset) );
                if ( from_ifsymbol )
                    {
                    labelx = startx + pix_to_ps( 5 );
                    labely = starty + pix_to_ps( 12 );
                    }
                break;
            case LEFT:

                /*
                 *  start drawing the line at the leftmost edge of the symbol
                 */

                startx = pix_to_ps( lineseg->start_x - dest->width/2 -
                    xoffset );
                starty = pix_to_ps( pix_centering_height() - (lineseg->start_y -
                    yoffset) );
                if ( from_ifsymbol )
                    {
                    labelx = startx - pix_to_ps( 40 );
                    labely = starty + pix_to_ps( 12 );
                    }
                break;
            case UP:

                /*
                 *  start drawing the line at the top edge of the symbol
                 */

                startx = pix_to_ps( lineseg->start_x - xoffset );
                starty = pix_to_ps( pix_centering_height() -
                    (lineseg->start_y - dest->height/2 - yoffset) );
                if ( from_ifsymbol )
                    {
                    labelx = startx + pix_to_ps( 10 );
                    labely = starty + pix_to_ps( 12 );
                    }
                break;
            case DOWN:

                /*
                 *  start drawing the line at the bottom edge of the symbol
                 */

                startx = pix_to_ps( lineseg->start_x - xoffset );
                starty = pix_to_ps( pix_centering_height() -
                    (lineseg->start_y + dest->height/2 - yoffset) );
                if ( from_ifsymbol )
                    {
                    labelx = startx + pix_to_ps( 10 );
                    labely = starty - pix_to_ps( 12 );
                    }
                break;
        }
        fprintf(fp, "%d %d moveto\n", startx, starty);

        /*
         *  loop through all the segs; if seg's arrowpt is non-zero, draw up
         *  to arrow and break, else draw seg and continue.
         */

        while ( lineseg )
            {
            if ( ( !(lineseg->arrow_x)) && (!(lineseg->arrow_y)) )
                {
                endx = pix_to_ps( lineseg->end_x - xoffset );
                endy = pix_to_ps( pix_centering_height() - (lineseg->end_y-yoffset));
                fprintf( fp, "%d %d lineto\n", endx, endy );
                }
            else
                {
                endx = pix_to_ps( lineseg->arrow_x - xoffset);
                endy = pix_to_ps( pix_centering_height()-(lineseg->arrow_y-yoffset));
                fprintf( fp, "%d %d lineto\n", endx, endy );
```

```
                fprintf( fp, "stroke\n" );
                print_arrows( endx, endy, lineseg->orientation, fp );
                break;
                }
            lineseg = (LineSeg *)lineseg->next;
            }

        /*
         *  print labels for line coming from if symbol
         */

        if ( from_ifsymbol )
            {
            fprintf( fp, "%d %d moveto\n", labelx, labely);
            fprintf( fp, "%s show\n",
                (line == dest->Sym.IfSym.true_line) ? "(TRUE)" : "(FALSE)" );
            fprintf( fp, "stroke\n" );
            from_ifsymbol = 0;
            }

        linelist = (LineList *)linelist->next;
    }       /*while*/
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  print_symbols()
*
* MODULE FUNCTION:
*
*   This routine prints all the symbols in the symbol_Map in postscript.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  print_symbols(fp, xoffset, yoffset )

    FILE *fp;
    int xoffset, yoffset;

{
    int     pswidth, psheight, psulcx, psulcy, i, psx, psy, midx, midy;
    char    result[300];

    /*
     *  go thru Symbol_Map; for each sym, put coordinates, size, and function
     *  call on postscript stack
     */

    for (i=0; i<MAX_SYMBOLS; i++)
        {
        if (Symbol_Map[i].symbol_type != NONE)
            {

            /*
             *  compute midpt and ulc of symbol in postscript units.
             *  offsets allow us to place the drawing anywhere on the page.
             */

            midx = Symbol_Map[i].ulcx +
                    Symbol_Map[i].width/2 - (xoffset-pix_left_margin());

            midy =  Symbol_Map[i].ulcy + Symbol_Map[i].height/2 - yoffset;

            psx = pix_to_ps( midx );
            psy = pix_to_ps( pix_centering_height() - midy );

            psulcx = pix_to_ps( Symbol_Map[i].ulcx - (xoffset-pix_left_margin()) );

            psulcy = pix_to_ps(pix_centering_height() - (Symbol_Map[i].ulcy - yoffset) );

            pswidth = pix_to_ps(Symbol_Map[i].width);
            psheight = pix_to_ps(Symbol_Map[i].height);

            switch( Symbol_Map[i].symbol_type )
                {
                case BEGIN:
                    fprintf( fp, "%d %d %d ellipse\n", psx, psy, pswidth/2 );
                    break;

                case END:
                    fprintf(fp, "%d %d %d circle\n", psx, psy, pswidth/2 );
```

```
        break;

    case SET:

        /*
         *  change ps font to symbol's font only if we are not in
         *  reduced mode; else use teeny font that is set in
         *  make_ps_file.
         */

        if ( PrintScale == NORMAL )
            {
            if (Symbol_Map[i].font == small_font)
                fprintf(fp, "%d SetScale\n", 1);
            else
                fprintf(fp, "%d SetScale\n", 0);
            }

        /*
         *  if we are not printing a report, use the toggle to
         *  determine what text to print.
         */

        if ( !Report )
            cvt_to_str_array( LogOrCompText ?
                Symbol_Map[i].Sym.IfSym.comp_expr :
                Symbol_Map[i].Sym.IfSym.logical_expr,
                result);

        /*
         *  else print log text if there
         */

        else if (Symbol_Map[i].Sym.IfSym.logical_expr)
            cvt_to_str_array(Symbol_Map[i].Sym.IfSym.logical_expr, result);
        else

        /*
         *  else print comp text
         */

            cvt_to_str_array(Symbol_Map[i].Sym.IfSym.comp_expr, result);

        fprintf( fp, "%d %d %d %d %s set\n", psulcx, psulcy,
            psheight, pswidth,  result);

        /*
         *  if we are not in reduced mode, change text back to default.
         *  reduced mode uses only a single, teeny font.
         */

        if ( PrintScale == NORMAL )
            fprintf(fp, "%d SetScale\n", 1);
        break;

    case PAUSE:
        fprintf( fp, "%d %d %d (%s) pause\n",
            psx, psy, pswidth/2, Symbol_Map[i].text );
        break;

    case GOTO:
        fprintf(fp, "%d %d %d %d (%s) goto\n",
            psulcx, psulcy, psheight, pswidth, Symbol_Map[i].text);
        break;

    case START:
        fprintf( fp, "%d %d %d %d (%s) sstart\n", psulcx, psulcy,
            psheight, pswidth, Symbol_Map[i].text);
        break;

    case PRINT:

        /*
         *  change ps font to symbol's font only if we are not in
         *  reduced mode; else use teeny font that is set in
         *  make_ps_file.
         */

        if ( PrintScale == NORMAL )
            {
            if (Symbol_Map[i].font == small_font)
                fprintf(fp, "%d SetScale\n", 1);
            else
                fprintf(fp, "%d SetScale\n", 0);
            }

        cvt_to_str_array(Symbol_Map[i].text, result);
        fprintf( fp, "%d %d %d %d %d %d %s pprint\n", psulcx, psulcy,
            psheight, pswidth, pix_to_ps(23), pix_to_ps(56), result);

        /*
         *  if we are not in reduced mode, change text back to default.
         *  reduced mode uses only a single, teeny font.
         */

        if ( PrintScale == NORMAL )
            fprintf(fp, "%d SetScale\n", 1);
        break;

    case STOP:
        fprintf( fp, "%d %d %d %d (%s) sstop\n", psulcx, psulcy,
            pix_to_ps(35), pix_to_ps(25), Symbol_Map[i].text);
        break;

    case TEXT:
        if ( PrintScale == NORMAL )
            {
            if (Symbol_Map[i].font == small_font)
                fprintf(fp, "%d SetScale\n", 1);
            else
                fprintf(fp, "%d SetScale\n", 0);
            }
        cvt_to_str_array(Symbol_Map[i].text, result);
        fprintf( fp, "%d %d %s text\n", psulcx, psulcy, result);
        if ( PrintScale == NORMAL )
            fprintf(fp, "%d SetScale\n", 1);
        break;

    case IF:

        /*
         *  same as set
         */

        if ( PrintScale == NORMAL )
            {
            if (Symbol_Map[i].font == small_font)
                fprintf(fp, "%d SetScale\n", 1);
```

```
                else
                    fprintf(fp, "%d SetScale\n", 0);
                }

            if ( !Report )
                cvt_to_str_array( LogOrCompText ?
                    Symbol_Map[i].Sym.IfSym.comp_expr :
                    Symbol_Map[i].Sym.IfSym.logical_expr,
                    result);
            else if (Symbol_Map[i].Sym.IfSym.logical_expr)
                cvt_to_str_array(Symbol_Map[i].Sym.IfSym.logical_expr, result);
            else
                cvt_to_str_array(Symbol_Map[i].Sym.IfSym.comp_expr, result);
            fprintf(fp, "%d %d %d %d %d %s 1if\n",
                psulcx, psulcy, pswidth, psheight, pix_to_ps(23), result );

            if ( PrintScale == NORMAL )
                fprintf(fp, "%d SetScale\n", 1);
            break;
        }   /*switch*/

    if (Symbol_Map[i].from)
        print_from_lines( Symbol_Map[i].from, fp, pswidth, psheight,
            (xoffset-pix_left_margin()), yoffset );

    }   /*if*/
}       /*for*/
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  set_arrows()
*
* MODULE FUNCTION:
*
*   This routine  sets the proper endpts of a line to show the arrows.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  set_arrows(lineseg)

    LineSeg     *lineseg;

{
    /*
     *  depending on orientation, set the last pt of a segment to be just
     *  outside the symbol it connects to.
     */

    switch (lineseg->orientation)
        {
        case UP:
            lineseg->arrow_x = lineseg->end_x;
            lineseg->arrow_y = LDendPtr->ulcy + LDendPtr->height - 1;
            break;

        case DOWN:
            lineseg->arrow_x = lineseg->end_x;
            lineseg->arrow_y = LDendPtr->ulcy - 1;
            break;

        case RIGHT:
            lineseg->arrow_y = lineseg->end_y;
            lineseg->arrow_x = LDendPtr->ulcx - 1;
            break;

        case LEFT:
            lineseg->arrow_y = lineseg->end_y;
            lineseg->arrow_x = LDendPtr->ulcx + LDendPtr->width - 1;
            break;

        }
    draw_arrows(lineseg, FALSE);
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  zoom()
*
* MODULE FUNCTION:
*
*   This routine  zooms in or out.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  zoom(direction)

    int     direction;

{
    int                 n, failed = FALSE;
    extern void         set_null_undo_event();
    XmString            zoom_string;
    Widget              hscroll, vscroll;
    Arg                 args[8];
    Font                sm_font, lg_font;
    GContext            gcon;
    XFontStruct         *fs;

    /*
     *  set whether we are zooming in or out
     */

    if ( direction == OUT )
        Zoomed = TRUE;
    else
        Zoomed = FALSE;

    /*
     *  Clear all cells so we can move all the symbols to their new
     *  locations without fear of collision with other symbols or lines.
     */

    init_cell_map();

    /*
     *  Clear all the lines and labels from the work area. This is faster
     *  and neater - you don't see lines and labels disappearing 1 by 1.
     */

    XClearWindow( display, XtWindow(draw_area) );

    /*
     *  Must redraw symbols and labels in small font
     */

    gcon = XGContextFromGC(LDgc);
    fs = XQueryFont(display, gcon);

    if (direction == OUT)
        {
        sm_font = XLoadFont(display, "6x10");
        XSetFont(display, LDgc,  (Font)sm_font);
        XSetFont(display, WAgc,  (Font)sm_font);
        }
    else
        {
        lg_font = XLoadFont(display, "fixed");
        XSetFont(display, LDgc,  (Font)lg_font);
        XSetFont(display, WAgc,  (Font)lg_font);
        }

    /*
     *  if we can't redraw symbol or a line, prepare to reverse zoom.
     */

    if ( !zoom_symbols(direction))
        failed = TRUE;

    if ( !zoom_lines(direction))
        failed = TRUE;

    /*
     *  Change popup menu's zoom string; unmap scrollbars when zoomed out.
     */

    XtSetArg(args[0], XmNlabelString, &zoom_string);
    XtGetValues(menu_list[2], args, 1);
    XmStringFree(zoom_string);

    XtSetArg( args[0], XmNverticalScrollBar, &vscroll );
    XtSetArg( args[1], XmNhorizontalScrollBar, &hscroll );
    XtGetValues(scr_WA, args, 2);

    /*
     *  reset popup menu string to zoomed [in | out]
     *  and [un]map scroll bars
     */

    if (direction == OUT)
        {
        zoom_string = XmStringCreate( "Zoom In",   XmSTRING_DEFAULT_CHARSET );
        XtUnmapWidget(hscroll);
        XtUnmapWidget(vscroll);
        }
    else
        {
        zoom_string = XmStringCreate( "Zoom Out",   XmSTRING_DEFAULT_CHARSET );
        XtMapWidget(hscroll);
        XtMapWidget(vscroll);

        /*
         *  make sure that as symbols are redrawn, they don't cover the scrollbars.
         */

        XRaiseWindow(display, XtWindow(hscroll));
        XRaiseWindow(display, XtWindow(vscroll));
        }


    XtSetArg( args[0], XmNlabelString, zoom_string );
    XtSetValues( menu_list[3], args, 1);
    XmStringFree( zoom_string );

    /*
     *  Make sure the menu buttons are nice and big.
```

```
     */

    XtSetArg( args[0], XmNwidth, MIN_BTN_WIDTH );
    XtSetValues( menu_list[3], args, 1 );

    /*
     *  can't undo in 'zoomed out' mode what was last done in 'zoomed in'
     *  mode (and vice versa), so we tell undo not to try if the user next selects undo.
     */

    set_null_undo_event();

    if (failed)
        {

    /*
     *  zoom failed because of inability to scale fonts in Motif; zoom back in/out.
     */

        if ( direction == OUT )
            zoom( IN );
        else
            zoom( OUT );
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  zoom_lines()
*
* MODULE FUNCTION:
*
*   This routine changes all the lines in the draw area, then redraws them in their
*   new size.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  zoom_lines(direction)

    int         direction;

{
    int         failed = 1, i;
    LineList    *from_line, *next_line;
    LineSeg     *line_seg;

    for (i=0; i<MAX_SYMBOLS; i++)
        if (Symbol_Map[i].symbol_type != NONE)
            {

            /*
             *  get list of lines entering this symbol
             */

            if (Symbol_Map[i].from)
                {
                LDendPtr = &Symbol_Map[i];
                from_line = (LineList *)Symbol_Map[i].from;
                while (from_line)
                    {
                    next_line = (LineList *)from_line->next;

                    /*
                     *  change endpts of linesegs of lines entering symbol
                     */

                    change_line((Line *)from_line->line, direction);
                    if ( !check_zoomed_line(from_line->line) )
                        {
                        user_ack("zoom failed; notify developers");
                        failed = 0;
                        }

                    /*
                     *  set cell map for all the segs in this line
                     */

                     line_seg = from_line->line->line;
                     while (line_seg)
                        {
                        set_cell_map_line(line_seg, from_line->line);
                        line_seg = (LineSeg *)line_seg->next;
                        }
```

```
                from_line = (LineList *)next_line;
                }
            }
        }
    return failed;
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  zoom_symbols()
*
* MODULE FUNCTION:
*
*   This routine changes all the symbols in the draw area, then redraws them in their
*   new size.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  zoom_symbols( direction )

    int     direction;

{
    int                 failed = 1, n = 0, i;
    Arg                 args[8];
    XWindowAttributes   attribs;

    for ( i=0; i<MAX_SYMBOLS; i++ )
        {

        /*
         *  for each symbol, redraw symbol in smaller size and font,
         *  give up old size's and position's real estate, and claim new real estate.
         */

        if ( Symbol_Map[i].symbol_type != NONE )
            {
            XClearWindow( display, XtWindow(Symbol_Map[i].mycanvas) );

            if (!XGetWindowAttributes(display,XtWindow(Symbol_Map[i].mycanvas),&attribs))
                error_handler( ERR_SYM_ATTRIBS, "zoom_symbols" );

            /*
             *  shrink symbol, reposition symbol to 1/2 its previous x and y values
             */

            n = 0;
            XtSetArg( args[n], XmNheight, op(attribs.height, direction) ); n++;
            XtSetArg( args[n], XmNwidth, op(attribs.width, direction) ); n++;
            XtSetArg( args[n], XmNx, op(attribs.x, direction) ); n++;
            XtSetArg( args[n], XmNy, op(attribs.y, direction) ); n++;
            XtSetValues( Symbol_Map[i].mycanvas, args, n );

            Symbol_Map[i].ulcx = op( attribs.x, direction );
            Symbol_Map[i].ulcy = op( attribs.y, direction );

            /*
             *  all zoomed symbols are drawn with a teeny font; normally they may
             *  have different fonts.
             */

            if ( direction == OUT )
                redraw_sym_num( i, teeny_font );
            else
```

```
            redraw_sym_num( i, Symbol_Map[i].font );

        /*
         *  lower the symbol so it doesn't obscure the scroll bars.
         */

        XLowerWindow( display, XtWindow(Symbol_Map[i].mycanvas) );

        /*
         *  set Symbol_Map entry fields
         */

        update_pos_fields( &Symbol_Map[i], Symbol_Map[i].mycanvas,
                           Symbol_Map[i].ulcx, Symbol_Map[i].ulcy );

        /*
         *  new claim can fail because of inability to scale fonts in Motif;
         *  if this happens, notify user
         */

        if ( !(set_cell_map_sym( SYMBOL_CELL, &Symbol_Map[i], Symbol_Map[i].cell_x,
            Symbol_Map[i].cell_y, Symbol_Map[i].cell_width,
            Symbol_Map[i].cell_height)) )
                {
                user_ack("zoom failed; notify developers");
                failed = 0;
                }
        }
    }
return failed;
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_palette()
*
* MODULE FUNCTION:
*
*   This routine handles events when the user pushes a BEGIN or END palette
*   button.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/cursorfont.h>
#include <Xm/Xm.h>

#include "gcb.h"
#include "widgets.h"
#include "fonts.h"


XtCallbackProc  cbr_palette( w, type, unused )

    Widget  w;
    int     type;
    caddr_t unused;

{
    XWindowAttributes   wattribs;
    int                 n, ht, wid;
    Arg                 args[10];


    /*
     *  The user is already in AddSymbol mode, they must want to cancel the
     *  adding of a new symbol.
     */

    if ( Mode == AddSymbol )
        {
        cancel_draw();
        return;
        }

    /*
     *   we are in help mode; popup help specific to the BEGIN or END.
     */

    if ( Mode == Help )
        {
        cbr_help( w, type, unused );
        return;
        }

    /*
     *  If this event hasn't already been handled and we aren't in EditMode, then
     *  ignore this event.
     */

    if ( Mode != EditSymbol )
        return;

    /*
     *  Make sure we have a valid element, can't add a symbol if we don't
     *  have a valid element.
     */

    if (! ValidElement )
        {
        error_handler( NO_ELEMENT, NULL );
        return;
        }

    /*
     *  Record which item the user last selected. This is used to pop down whatever
     *  popup results from the user's choice.
     */

    invert( type );

    /*
     *  See if we can get data structures for another symbol.
     */

    if ( next_avail_sym() < 0 )
        {
        user_ack("Fatal Error: couldn't get symbol structure");
        elog(1,"Fatal Error: couldn't get symbol structure");
        return;
        }

    /*
     *  only 1 Begin symbol per element
     */

    if ( (type == BEGIN) && (Begin_Sym) && (Begin_Sym != &Symbol_Map[nextsymbol]) )
        {
        elog(3,"install failed: nextsymbol = %d\n", nextsymbol);
        user_ack("The GCB grammar definition allows only 1 BEGIN symbol per Element");
        return;
        }

    /*
     *  retrieve dimensions of symbol window.
     */

     if (! XGetWindowAttributes(display,XtWindow(w),&wattribs) )
        error_handler( ERR_SYM_ATTRIBS, "cbr_palette" );

    /*
     *  adjust the dimensions of the symbol according to its type.
     *  if we are zoomed, set the dimensions accordingly.
     */

    ht  = (type==BEGIN) ? wattribs.height-15 : wattribs.height;
    wid = (type==BEGIN) ? wattribs.width     : wattribs.width+5;

    n = 0;
    if ( Zoomed )
        {
        XtSetArg( args[n], XmNheight, op(ht, 2) );  n++;
        XtSetArg( args[n], XmNwidth,  op(wid, 2) ); n++;
```

```
        }
    else
        {
        XtSetArg( args[n], XmNheight, ht ); n++;
        XtSetArg( args[n], XmNwidth, wid ); n++;
        }

    /*
     *  Store the type of the symbol in its XmNuserdata resource.
     */

    XtSetArg( args[n], XmNuserData, type ); n++;
    XtSetValues( current_symbol, args, n );

    Mode = AddSymbol;
    upd_mode_panel();

    /*
     *  Pass the type, current sym #, Graphics context, and font to the
     *  routine which draws the symbol.
     */

    draw_symbol( type, current_symbol, WAgc, small_font );
}
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    cbr_printset.c
*
* FILE FUNCTION:
*
*   This file contains the routines which respond to PRINT, TEXT, GOTO, START, and
*   STOP button press events.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /home/project/3531/Docu/GCB.spec.doc
*
*
* FILE MODULES:
*
*   build_call_popup()      - builds the goto, activate, and stop popups.
*   build_pause_popup()     - builds the pause popup.
*   build_start_popup()     - builds the ACTIVATE/STOP popup.
*   build_text_popup()      - builds the popup to input free text.
*   cbr_pause_units()       - Reprints user's choice of pause units in the value text.
*   cbr_pause_value()       - Reprints user's choice of pause value in the value text.
*   cbr_printset()          - Responds to PRINT, TEXT, GOTO, START, and STOP buttons
*   cbr_startstop_sel()     - Handles button press events in element selection list.
*
*****************************************<----->*****************************************/

#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>

#include "gcb.h"
#include "cbr.h"
#include "widgets.h"
#include "element_file.h"
#include "constants.h"

/*
 *  header strings for popups.
 */

char    goto_string[] = "Select from the list the name of the Element to CALL\n or enter
 a new Element name. Then click the Done button." ;
                  !
char    start_string[] = "Select from the list the name of the Comp or enter\n a new Com
p name. Then click the Done button.";
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  build_call_popup()
*
* MODULE FUNCTION:
*
*   This routine builds the CALL popup.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  build_call_popup( parent )

    Widget parent;

{
    Arg args[2];


    dlg_call = cr_popup( NULLS, parent, NULLS );

    FormW = cr_form( NULLS, dlg_call, NULL, NULL );
    set_attribs( FORM, FormW, 425, 400, XmRESIZE_NONE );

    lbl_call_header = cr_label( NULLS, FormW, goto_string, 0, 3, 11, 5, 95);
    cr_label( NULLS, FormW, "Element Type:", 0, 18, 21, 9, IGNORE);

    rb_call = cr_radio_box( NULLS, FormW, XmHORIZONTAL );

    tgl_call_ce = cr_toggle( NULLS, rb_call, "Comp Element", NULL, 0, 0);
    XtSetArg( args[0], XmNuserData, ELEMENT );
    XtSetValues( tgl_call_ce, args, 1 );

    tgl_call_le =cr_toggle( NULLS, rb_call, "Library Element", NULL, 0,0);
    XtSetArg( args[0], XmNuserData, LIB );
    XtSetValues( tgl_call_le, args, 1 );

    arm_tgl( tgl_call_ce );

    cr_label( NULLS, FormW, "Element Name:", 0, 25, 28, 9, IGNORE);
    cr_label( NULLS, FormW, "Element List:", 0, 32, 35, 9, IGNORE);
    txt_call_ne   = cr_text ( NULLS, FormW, NULL, NULL, NULLS, FALSE, 1, 35);

    list_elem = cr_list( NULLS, FormW, 10, 215 );
    set_attribs( LIST, list_elem, XmSINGLE_SELECT, NULL, NULL );

    XtAddCallback( list_elem,  XmNsingleSelectionCallback, cbr_startstop_sel, ELEMENT );
    XtAddCallback( tgl_call_le, XmNarmCallback, cbr_element_type, list_elem );
    XtAddCallback( tgl_call_ce, XmNarmCallback, cbr_element_type, list_elem );

    cr_separator( NULLS, FormW, 84, IGNORE, 1, 99 );

    CancelW = cr_command( NULLS, FormW, "Cancel", cbr_cancel,1 );
    DoneW   = cr_command( NULLS, FormW, "Done",   cbr_done,  1 );
    HelpW   = cr_command( NULLS, FormW, "Help",   cbr_help,  CALL_ELEM );

    set_position( rb_call,              15, IGNORE, 35, IGNORE );
    set_position( txt_call_ne,          24, IGNORE, 35, IGNORE );
```

```
    set_position( XtParent(list_elem),   33, IGNORE, 35, IGNORE );
    set_position( CancelW,               91, IGNORE,  5, IGNORE );
    set_position( DoneW,                 91, IGNORE, 40, IGNORE );
    set_position( HelpW,                 91, IGNORE, 75, IGNORE );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  build_pause_popup()
*
* MODULE FUNCTION:
*
*   This routine builds the pause popup.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  build_pause_popup( parent )

    Widget parent;

{
    dlg_pause = cr_popup( NULLS, parent, "Pause Input" );
    FormW = cr_form( NULLS, dlg_pause, NULL, NULL );
    set_attribs( FORM, FormW, 300, 380, XmRESIZE_NONE );

    cr_label( NULLS, FormW, "Value", 0, 10, IGNORE, 20, IGNORE );
    txt_pause_value = cr_text  ( NULLS, FormW, NULL, NULL, NULLS, FALSE, 1, 15 );
    rc_pause_num    = cr_rowcol( NULLS, FormW, 4, XmHORIZONTAL, NULL, NULL );

    BtnW = cr_command( NULLS, rc_pause_num, " 7 ", cbr_pause_value, "7" );
    BtnW = cr_command( NULLS, rc_pause_num, " 8 ", cbr_pause_value, "8" );
    BtnW = cr_command( NULLS, rc_pause_num, " 9 ", cbr_pause_value, "9" );
    BtnW = cr_command( NULLS, rc_pause_num, " 4 ", cbr_pause_value, "4" );
    BtnW = cr_command( NULLS, rc_pause_num, " 5 ", cbr_pause_value, "5" );
    BtnW = cr_command( NULLS, rc_pause_num, " 6 ", cbr_pause_value, "6" );
    BtnW = cr_command( NULLS, rc_pause_num, " 1 ", cbr_pause_value, "1" );
    BtnW = cr_command( NULLS, rc_pause_num, " 2 ", cbr_pause_value, "2" );
    BtnW = cr_command( NULLS, rc_pause_num, " 3 ", cbr_pause_value, "3" );
    BtnW = cr_command( NULLS, rc_pause_num, " 0 ", cbr_pause_value, "0" );

    rc_pause_units = cr_rowcol( NULLS, FormW, 1, XmVERTICAL, NULL, NULL );

    BtnW = cr_command( NULLS, rc_pause_units, "ms", cbr_pause_units,  "ms" );
    BtnW = cr_command( NULLS, rc_pause_units, "sec",cbr_pause_units, "sec" );
    BtnW = cr_command( NULLS, rc_pause_units, "min",cbr_pause_units, "min" );

    cr_separator( NULLS, FormW, 75, IGNORE, 1, 99 );

    CancelW         = cr_command( NULLS, FormW, "Cancel", cbr_cancel, 1 );
    BtnW            = cr_command( NULLS, FormW, "Done",   cbr_done,   1 );
    HelpW           = cr_command( NULLS, FormW, "Help",   cbr_help,   PAUSE_ELEM );

    set_position( txt_pause_value,  10, IGNORE, 40, IGNORE );
    set_position( rc_pause_num,     30, IGNORE, 20, IGNORE );
    set_position( rc_pause_units,   30, IGNORE, 70, IGNORE );
    set_position( CancelW,          85, IGNORE,  3, IGNORE );
    set_position( BtnW,             85, IGNORE, 36, IGNORE );
    set_position( HelpW,            85, IGNORE, 68, IGNORE );
}
```

```
/*****************************************<-----*****************************************
*
* MODULE NAME:  build_start_popup()
*
* MODULE FUNCTION:
*
*   This routine builds the ACTIVATE and STOP popup.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<-----*****************************************/

void  build_start_popup( parent )

    Widget parent;

{
    dlg_start = cr_popup( NULLS, parent, NULLS );

    FormW = cr_form( NULLS, dlg_start, NULL, NULL );
    set_attribs( FORM, FormW, 425, 400, XmRESIZE_NONE );

    lbl_start_hdr = cr_label( NULLS, FormW, start_string, 0, 3, 11, 5, 95);

    cr_label( NULLS, FormW, "Comp Name:", 0, 23, IGNORE, 9, IGNORE);
    cr_label( NULLS, FormW, "Comp List:", 0, 32, IGNORE, 9, IGNORE);

    txt_start  = cr_text( NULLS, FormW, NULL, NULL, NULLS, FALSE, 1, 35);

    list_comp = cr_list( NULLS, FormW, 10, 215 );
    set_attribs( LIST, list_comp, XmSINGLE_SELECT, NULL, NULL );

    XtAddCallback( list_comp, XmNsingleSelectionCallback, cbr_startstop_sel, COMP );

    cr_separator( NULLS, FormW, 84, IGNORE, 1, 99 );

    CancelW = cr_command( NULLS, FormW, "Cancel", cbr_cancel,1 );
    DoneW   = cr_command( NULLS, FormW, "Done",   cbr_done,  1 );
    HelpW   = cr_command( NULLS, FormW, "Help",   cbr_help,  ACTIVATE_COMP );

    set_position( txt_start,            22, IGNORE, 35, IGNORE );
    set_position( XtParent(list_comp),  33, IGNORE, 35, IGNORE );
    set_position( CancelW,              91, IGNORE,  5, IGNORE );
    set_position( DoneW,                91, IGNORE, 40, IGNORE );
    set_position( HelpW,                91, IGNORE, 75, IGNORE );
}
```

```
/*****************************************<-----*****************************************
*
* MODULE NAME:  build_text_popup()
*
* MODULE FUNCTION:
*
*   This routine builds the popup to input free text.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<-----*****************************************/

void  build_text_popup(parent)

    Widget parent;

{
    int         n;
    Arg         args[2];
    char        *text_string,
                *print_string;

    dlg_text = cr_popup( NULLS, parent, NULLS );
    XtSetArg( args[0], XmNdialogStyle, XmDIALOG_MODELESS );
    XtSetValues( dlg_text, args, 1 );

    FormW = cr_form( NULLS, dlg_text, NULL, NULL );
    set_attribs( FORM, FormW, 450, 350, XmRESIZE_NONE );

    text_string = (char *) malloc( sizeof(char)*(41) );
    strcat( text_string, "Input the text.\n" );
    strcat( text_string, "Then click the Done button." );

    print_string = (char *) malloc( sizeof(char)*(61) );
    strcat( print_string, "Input the expression to be printed.\n" );
    strcat( print_string, "Then click the Done button." );

    text_header  = XmStringLtoRCreate( (char *) text_string,  XmSTRING_DEFAULT_CHARSET);
    print_header = XmStringLtoRCreate( (char *) print_string, XmSTRING_DEFAULT_CHARSET);

    lbl_text_header = cr_label( NULLS, FormW, NULLS, 0, 6, 14, 5, 95 );

    scr_text = cr_scr_text( NULLS, FormW, 10, True, 375, 0, 0 );
    XtSetArg( args[0], XmNscrollingPolicy, XmAUTOMATIC );
    XtSetValues( scr_text, args, 1 );

    BtnW            = cr_command( NULLS, FormW, "Attributes", cbr_done, 0 );
    DoneW           = cr_command( NULLS, FormW, "Done",       cbr_done, 1 );
    CancelW         = cr_command( NULLS, FormW, "Cancel",     cbr_cancel, 1 );
    HelpW           = cr_command( NULLS, FormW, "Help",       cbr_help,TEXT_POPUP);

    cr_separator( NULLS, FormW, 80, IGNORE, 1, 99 );

    set_position( XtParent(scr_text),   20, IGNORE, 10, IGNORE );
    set_position( CancelW,              88, IGNORE,  3, IGNORE );
    set_position( BtnW,                 88, IGNORE, 27, IGNORE );
    set_position( DoneW,                88, IGNORE, 53, IGNORE );
    set_position( HelpW,                88, IGNORE, 77, IGNORE );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_pause_units()
*
* MODULE FUNCTION:
*
*   This routine reprints the user's choice of pause units in the value text.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_pause_units( w, client_data, call_data )

    Widget w;
    caddr_t client_data;
    caddr_t call_data;

{
    /*
     *  retrieve text string from units button, set value text to value concatted
     *  with units.
     */

    XmTextSetString( txt_pause_value,
        strcat(XmTextGetString(txt_pause_value), (char *)" ") );
    XmTextSetString( txt_pause_value,
        strcat(XmTextGetString(txt_pause_value), (char *)client_data) );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_pause_value()
*
* MODULE FUNCTION:
*
*   This routine reprints the user's choice of pause value in the value text.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_pause_value( w, client_data, call_data )

    Widget w;
    caddr_t client_data;
    caddr_t call_data;

{
    /*
     *  retrieve string from value button; set value string to this string.
     */

    XmTextSetString( txt_pause_value,
        strcat( XmTextGetString(txt_pause_value), (char *)client_data) );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_printset()
*
* MODULE FUNCTION:
*
*   This routine responds to the users clicking on a PRINT, TEXT, GOTO, START, or STOP
*   button.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_printset( w, item, unused )

    Widget  w;
    int     item;
    caddr_t unused;

{
    Arg                 args[10],
                        header_args[1];
    XWindowAttributes   wattribs;
    int                 n, ht, wid;
    char                tmpStr[MAX_NAME];

    /*
     *  cancel palette op by pressing any palette item again
     */

    if ( Mode == AddSymbol )
        {
        cancel_draw();
        return;
        }

    /*
     *  popup help specific to this palette item.
     */

    if ( Mode == Help )
        {
        cbr_help( w, item, unused );
        return;
        }

    if ( Mode != EditSymbol )
        return;

    /*
     *  Make sure we have a valid element, can't add a symbol if we don't
     *  have a valid element.
     */

    if (! ValidElement )
        {
        error_handler( NO_ELEMENT, NULL );
        return;
        }
```

```
/*
 *  Currently, the GCB does not allow the user to add symbols which affect
 *  the Comp symbol table to be added to a Library Element.
 */

if ( ElementType == LIB )
    switch ( item )
        {
        case GOTO  :
        case STOP  :
        case START :
        case PAUSE :

            error_handler( LIB_ELEM_SYM, NULL );
            return;
        }

/*
 *  Store the id of the last selected item
 */

invert( item );

/*
 *  See if we have data structures for another symbol.
 */

if ( (next_avail_sym()) < 0 )
    return ;


Mode = AddSymbol;
upd_mode_panel();


if (! XGetWindowAttributes(display,XtWindow(w),&wattribs) )
    error_handler( ERR_SYM_ATTRIBS, "cbr_printset" );

/*
 *  Depending on type of symbol being created, set label for popup,
 *  size of symbol, and load the element list, if applicable.
 */

n = 0;
switch( item )
    {
    case GOTO:

        /*
         *  in the case of GOTO, set the Symbol_Map height to the
         *  default height of the widget, and the width to 37 pixels
         *  wider than the default.
         */

        ht = wattribs.height;
        wid = wattribs.width + 37;

        ElementListType = ELEMENT;

        /*
         *  get list of elements from the symbol table
         */
```

```
        if (! load_element_list(ELEMENT,list_elem) )
            {
            sscanf( selList[0], "%s", tmpStr );
            XmTextSetString( txt_call_ne, tmpStr );

            /*
             *   set default toggles
             */

               arm_tgl( tgl_call_ce );
            disarm_tgl( tgl_call_le );
            XtManageChild( dlg_call );
            }
        break;

    case STOP:

        /*
         *  in the case of STOP, set the Symbol_Map height to the
         *  default height of the widget + 22, and the width to 90 pixels.
         */

        ht = wattribs.height + 22;
        wid = 90;


        /*
         *  Load the list of Comp names for the current position into
         *  the selection list.
         */

        if (! load_element_list(COMP,list_comp) )
            {
            XmTextSetString( txt_start, selList[0] );
            XtManageChild( dlg_start );
            }
        break;

    case START:

        /*
         *  in the case of START, set the Symbol_Map height to the
         *  default height of the widget, and the width to 37 pixels
         *  wider than the default.
         */

        ht = wattribs.height;
        wid = wattribs.width + 37;


        if (! load_element_list(COMP,list_comp) )
            {
            XmTextSetString( txt_start, selList[0] );
            XtManageChild( dlg_start );
            }
        break;

    case PAUSE:
        ht = wattribs.height;
        wid = wattribs.width + 5;
        XtManageChild( dlg_pause );
        break;
    case TEXT:
        XtSetArg( header_args[0], XmNlabelString, text_header );
```

```
            XtSetValues( lbl_text_header, header_args, 1 );
            XtManageChild( dlg_text );
            break;
        case PRINT:
            ht = wattribs.height;
            wid = wattribs.width + 74;

            XtSetArg( header_args[0], XmNlabelString, print_header );
            XtSetValues( lbl_text_header, header_args, 1 );
            XtManageChild( dlg_text );
            break;
        }

    XtSetArg( args[n], XmNheight, (Zoomed) ? op(ht, 2)  : ht );  n++;
    XtSetArg( args[n], XmNwidth,  (Zoomed) ? op(wid, 2) : wid ); n++;

    XtSetArg( args[n], XmNuserData, item ); n++;
    XtSetValues( current_symbol, args, n );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_startstop_sel()
*
* MODULE FUNCTION:
*
*   This routine handles events when the user pushes a mouse button over an
*   item in the Element or Comp selection list.  This routine will extract
*   the text for the item selected by the user and will insert the text into
*   the single line text widget of the corresponding popup.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  cbr_startstop_sel( w, client_data, call_data )

    Widget                  w;
    int                     client_data;
    XmListCallbackStruct    *call_data;

{
    char *text,
         tmpStr[MAX_NAME];


    /*
     *  If we are working in the ACTIVATE or STOP popup, then get the item
     *  selected by the user and insert it into the text widget of the ACTIVATE/
     *  STOP popup.
     */

    if ( client_data == COMP )
        {
        XmStringGetLtoR( call_data->item, XmSTRING_DEFAULT_CHARSET, &text );
        XmTextSetString( txt_start, text );
        return;
        }

    /*
     *  Get the string/name which the user selected.  Scan out just the name
     *  portion, the string may have contained "Root Element" at the end.
     *  Insert the new string which contains just the element name into the
     *  text widget.
     */

    XmStringGetLtoR( call_data->item, XmSTRING_DEFAULT_CHARSET, &text );
    sscanf( text, "%s", tmpStr );
    XmTextSetString( txt_call_ne, tmpStr );
}
```

```
/****************************************<----->****************************************
*
* FILE NAME:    cbr_symbol.c
*
* FILE FUNCTION:
*
*   This file contains the routines which handle events in symbols.
*
* SPECIFICATION DOCUMENTS:
*
*   /home/project/3531/Docu/GCB.spec.doc
*
*
* FILE MODULES:
*
*   cbr_symbol()             - handles events when user pushes mouse button in symbol
*   cbr_symbol_move()        - handles motion and enter notify events in symbols.
*   create_new_sym()         - creates a new symbol when user pushes palette button.
*   handle_button_press()    - handles button press events in a symbol.
*   handle_button_release()  - handles button release events in a symbol.
*   set_origin()             - retrieves window's attributes and sets global origin.
*
****************************************<----->****************************************/

#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include <X11/cursorfont.h>
#include <X11/Shell.h>
#include <X11/StringDefs.h>

#include "gcb.h"
#include "cbr.h"
#include "widgets.h"
#include "lines.h"
#include "element_file.h"
#include "fonts.h"
#include "cursors.h"

/*
 *  id of most recently dragged symbol
 */

static Widget   dragged_sym;
```

```
/****************************************<----->****************************************
*
* MODULE NAME:  cbr_symbol()
*
* MODULE FUNCTION:
*
*   This routine handles events when the user pushes or releases a mouse button
*   while in a symbol area canvas.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
****************************************<----->****************************************/

XtCallbackProc  cbr_symbol( w, client_data, call_data)

    Widget                          w;
    caddr_t                         client_data;
    XmDrawingAreaCallbackStruct     *call_data;

{
    int                 sym_type;
    Arg                 args[3];
    XEvent              *event = (XEvent *)call_data->event;

    /*
     *  pass the event to handle_button_[press | release]
     */

    if ( event->type == ButtonPress )
        handle_button_press( w, client_data, call_data );
    else if ( event->type == ButtonRelease )
        handle_button_release( w, client_data, call_data );
}
```

```
/*************************************<----->*************************************
*
* MODULE NAME:  cbr_symbol_move()
*
* MODULE FUNCTION:
*
*   This routine handles motion and enter notify events in symbols.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*************************************<----->*************************************/

XtCallbackProc  cbr_symbol_move( w, client_data, event)

    Widget                          w;
    caddr_t                         client_data;
    XEvent                          *event;

{
    if  ( event->type == MotionNotify )
        {
        if ( (!(event->xmotion.state & Button1Mask)) && (w == current_symbol) &&
            (Mode == AddSymbol) )
            {

            /*
             *  user is moving symbol after selecting its palette button.
             *  move ulc of window to new location relative to event in window
             */

            XMoveWindow( display, XtWindow(w),
                (newx = (newx - (originx - event->xmotion.x))),
                (newy = (newy - (originy - event->xmotion.y))) );

            /*
             *  keep ruler bars up with moving symbol
             */

            XMoveWindow( display, XtWindow(v_rule), newx + originx + 2, 0 );
            XMoveWindow( display, XtWindow(h_rule), 0, newy + originy + 16 );
            }
        else if ( ( event->xmotion.state & Button1Mask) && ( Mode == DragSymbol ) &&
            ( dragged_sym == w ) )
            {

            /*
             *  user is dragging symbol.
             */

            XMoveWindow( display, XtWindow(w),
                (newx = (newx - (originx - event->xmotion.x))),
                (newy = (newy - (originy - event->xmotion.y))) );
            XMoveWindow( display, XtWindow(v_rule), newx + originx + 2, 0 );
            XMoveWindow( display, XtWindow(h_rule), 0, newy + originy + 16 );
            }
        }
    else if ( ( event->type == EnterNotify ) && ( w == current_symbol) &&
        ( Mode == AddSymbol) )
        {
        /*
         *  user has entered symbol after selecting its palette item. Set
         *  origin for future move events.
         */

        set_origin( w );
    }
```

```
/*****************************************<----->*****************************************
*
*MODULE NAME:  create_new_sym()
*
* MODULE FUNCTION:
*
*   This routine creates a new symbol when the user pushes a palette button.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

Widget  create_new_sym()

{
    Arg                     args[2];
    XSetWindowAttributes    swattribs;
    Widget                  symbol;
    extern XtCallbackRec    symbol_input_code[];

    /*
     *  create new symbol: don't map it yet ( map it only when the user brings
     *  the sprite into the canvas area - an EnterNotify event in cbr_canvas )
     */

    XtSetArg( args[0], XtNmappedWhenManaged, FALSE );
    XtSetArg( args[1], XmNinputCallback, symbol_input_code );
    symbol = ( Widget) XmCreateDrawingArea( draw_area, "symbol", args, 2 );

    if ( !(symbol) )

        /*
         *  Motif call failed, panic.
         */

        exit( ERR );
    XtManageChild( symbol );

    /*
     *  use workstation's backing store to repaint.
     */

    swattribs.backing_store = Always;
    XChangeWindowAttributes( display, XtWindow(symbol), CWBackingStore, &swattribs);

    XtAddEventHandler( symbol, ButtonMotionMask | EnterWindowMask | PointerMotionMask,
        FALSE, cbr_symbol_move, NULL );

    /*
     *  grab button events; else they go to symbol's parent, the draw area.
     */

    XGrabButton( display, AnyButton, AnyModifier, XtWindow(symbol), TRUE,
        ButtonPressMask | ButtonMotionMask | ButtonReleaseMask,
        GrabModeAsync, GrabModeAsync,
        XtWindow(symbol),
        None );

    return( symbol );
}
```

```
/*************************************<----->*************************************
*
* MODULE NAME:  handle_button_press()
*
* MODULE FUNCTION:
*
*   This routine handles button press events in a symbol.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*************************************<----->*************************************/

int  handle_button_press( w, client_data, call_data )

    Widget                          w;
    caddr_t                         client_data;
    XmDrawingAreaCallbackStruct     *call_data;

{
    Arg     args[1];
    XEvent  *event = (XEvent *) call_data->event;
    int     sym_type;

    if ( event->xbutton.button == Button1 )
        {
        elog(3,"symbol: button #%u down", event->xbutton.button);
        if ( (Mode == EditSymbol) &&
            (!(event->xbutton.state & ShiftMask)) &&        /*shift not down*/
            (!(event->xbutton.state & ControlMask)) )       /*control not down*/
            {

            /*
             *  user wants to drag this symbol; set variable to remember
             *  which symbol we are dragging; raise the symbol so we can
             *  drag it over others; and set the mode.
             */

            dragged_sym = w;
            XRaiseWindow( display, XtWindow(w) );
            Mode = DragSymbol;
            set_origin( w );
            }
        else if ( (w == current_symbol) && (Mode == AddSymbol) )
            {
            int i;
            XWindowAttributes   attribs;

            /*
             *  we are placing this symbol; find its current location.
             */

            if (! XGetWindowAttributes(display,XtWindow(w),&attribs) )
                error_handler( ERR_SYM_ATTRIBS, "handle_button_press" );

            /*
             *  try to install symbol; if successful, continue
             */

            XtSetArg( args[0], XmNuserData, &sym_type );
            XtGetValues( current_symbol, args, 1 );

            if ( (i = install_symbol(sym_type,
                            attribs.x,
                            attribs.y,
                            attribs.height,
                            attribs.width,
                            sym_text,
                            w,
                            WAFont)) >= 0 )
                {
                elog(3,"install successful");

                /*
                 *  save this event for undo
                 */

                set_symbol_event( Symbol_Map[i] );
                SaveNeeded = TRUE;
                if ( Audit )

                    /*
                     *  placing this symbol should make the element
                     *  incomplete.
                     */
                    audit( JUST_LINES );

                /*
                 * reset mode in mode panel
                 */

                Mode = EditSymbol;
                upd_mode_panel();
                }
                else

                    /*
                     *  install failed, allow user to try again.
                     */

                    return;
            }
        else if ( (Mode == EditSymbol) &&
            (event->xbutton.state & ShiftMask) )        /*shift down*/

            /*
             *  delete symbol
             */

            {
            remove_symbol( w );
            if ( Audit )
                audit( JUST_LINES );
            }
        else if ( (Mode == EditSymbol) &&
            (event->xbutton.state & ControlMask) )      /*control down*/
                {

                /*
                 *  audit/clear individual symbol
                 */

                if ( !Audited )
                    {
```

```
            elog(3,"auditing symbol");
            audit_symbol( w );
            Audited = 1;
            }
        else
            {
            elog(3,"clearing audited sym");
            clear_audit();
            Audited = 0;
            }
        }
    }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  handle_button_release()
*
* MODULE FUNCTION:
*
*   This routine handles button release events in a symbol.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int   handle_button_release( w, client_data, call_data)

    Widget                          w;
    caddr_t                         client_data;
    XmDrawingAreaCallbackStruct     *call_data;

{
    XEvent  *event = (XEvent *)call_data->event;
    int     cell_x,
            cell_y,
            true_x,
            true_y;


    elog(3,"symbol: button #%u up", event->xbutton.button);
    if ( (event->xbutton.button == Button1) && (Mode == DragSymbol) &&
         (w == dragged_sym))
        {

        /*
         *  update symbol position
         */

        elog(3,"updating sym pos");
        update_symbol_pos( w );
        Mode = EditSymbol;
        }
    else if ( event->xbutton.button == Button2 )
        {
        int sym;

            /*
             *  If we are in Edit Symbol mode, then the user wants to start
             *  line draw.  Determine which symbol we are in and init for line
             *  draw.
             */

            if ( Mode == EditSymbol )
                {

                /*
                 *  Determine which symbol we are in.
                 */

                for ( sym=0; sym<MAX_SYMBOLS; sym++ )
                    if ( Symbol_Map[sym].mycanvas == w )
                        {
```

```
            /*
             *  can't have line out of text canvas
             */

            if ( Symbol_Map[sym].symbol_type == TEXT )
                return;

            /*
             *  set global line start vars to midpt of this symbol
             */

            LDstartX = Symbol_Map[sym].ulcx + (Symbol_Map[sym].width/2);
            LDstartY = Symbol_Map[sym].ulcy +(Symbol_Map[sym].height/2);
            LDstartPtr = &(Symbol_Map[sym]);
            break;
            }

    if ( LDstartPtr->symbol_type == END )
        {

        /*
         *  can't have a line out of an end symbol.
         */

        user_ack("The end is the end.");
        return;
        }

    /*
     *  If this symbol is already fully connected, let the user
     *  know.
     */

    if ( LDstartPtr->symbol_type != IF )
        if ( LDstartPtr->next != NULL )
            {
            user_ack("Symbol already has a next symbol");
            return;
            }
        else
            LDside = NEXT_PTR;

    else
        {
        if ( (LDstartPtr->Sym.IfSym.true_line  != NULL) &&
            (LDstartPtr->Sym.IfSym.false_line != NULL) )
            {
            user_ack("Symbol has a TRUE and FALSE connector");
            return;
            }

        /*
         *  The user wants to build an IF line, determine which
         *  part he/she wants to connect.
         */

        else if ( (LDstartPtr->Sym.IfSym.true_line  == NULL) &&
                  (LDstartPtr->Sym.IfSym.false_line == NULL) )

                LDside = prompt_true_false( w );

        else
            if ( LDstartPtr->Sym.IfSym.true_line != NULL )
                LDside = FALSE_PTR;
            else
                LDside = TRUE_PTR;
            }

         /*
          *  Start building the "next line" structures.
          */

        start_Line( LDside );

        LDendX   = -1;
        LDendPtr = NULL;
        LDtype   = START_SEGMENT;
        Mode     = LineDraw;

        XDefineCursor( display, XtWindow(draw_area), tcross_cursor );
        XDefineCursor( display, XtWindow(draw_area), tcross_cursor );
        }

    /*
     *  User is in LineDraw mode and they want to terminate the
     *  current line in this symbol.
     */

    else if ( Mode == LineDraw )
        {

        if ( LDendX == -1 )
            {
            XBell( display, 0 );
            user_ack("Could not draw line, please try again to connect symbols");
            return;
            }

        /*
         *  Identify target symbol.
         */

        for ( sym=0; sym<MAX_SYMBOLS; sym++ )
            if ( Symbol_Map[sym].mycanvas == w)
                {
                if ( Symbol_Map[sym].symbol_type == TEXT )
                    {
                    XBell( display, 0 );
                    user_ack("Cannot connect a line to label text");
                    return;
                    }
                LDendPtr = &(Symbol_Map[sym]);
                break;
                }

        /*
         *  The events which draw the line may not have kept up with the
         *  mouse pointer.  Make sure the end of the line matches the
         *  actual mouse pointer location.
         */

        true_x = LDendPtr->ulcx + event->xbutton.x;
        true_y = LDendPtr->ulcy + event->xbutton.y;

        if ((LDendX != true_x) && (LDendY != true_y))
            {
            clear_line();
```

```
        if ( LDendX != LDstartX )
            LDendX = true_x;
        else
            LDendY = true_y;
        draw_line();
        }

    /*
     *  Make sure the line actually is within the symbol. The user may
     *  have placed the mouse in a symbol without the line actually
     *  making it into the symbol due to a delay in the processing of
     *  events.
     */

    cell_x = LDendX / CELL_SIZE;
    cell_y = LDendY / CELL_SIZE;

    if ( Cell_Map[cell_y][cell_x].cell_type != SYMBOL_CELL )
        {
        user_ack("End of line not in target symbol - setting anchor point");
        mid_Line();
        LDstartX = LDendX;
        LDstartY = LDendY;
        LDtype = MID_SEGMENT;
        LDendX  = -1;
        return;
        }

    /*
     *  Don't allow connect to same symbol.
     */

    if ( LDendPtr == LDstartPtr )
        {
        XBell( display, 0 );
        user_ack("Cannot connect to the same symbol");
        return;
        }

    /*
     *  Don't allow connect to Begin symbol.
     */

     if ( LDendPtr->symbol_type == BEGIN )
        {
        XBell( display, 0 );
        user_ack("Cannot connect to a BEGIN symbol");
        return;
        }

    /*
     *  Determine the "real" end X & Y to use for drawing the
     *  the line.  The line which was drawn will not extend
     *  into the end symbol because the mouse following
     *  routines are not fast enough.  Determine the real end
     *  X & Y, and redraw the line.
     */

    clear_line();
    if ( LDendX != LDstartX )
        LDendX = LDendPtr->ulcx+10;
    else
        LDendY = LDendPtr->ulcy+10;
                set_line();

                /*
                 *  Make sure the user has drawn a valid line.
                 */

                if ( valid_line(END_SEGMENT) == ERR )
                    {
                    XBell( display, 0 );
                    user_ack("Invalid connector - cannot go through a symbol");
                    return;
                    }

                /*
                 *  We have a valid end segment.
                 */

                LDtype = END_SEGMENT;
                end_Line();

                /*
                 *  if audit is on, audit element
                 */

                if ( Audit )
                    audit( JUST_LINES );

                /*
                 *  record this line event for undo
                 */

                set_line_event( LDlinePtr );

                /*
                 *  reset mode and cursor in main canvas and start symbol
                 */

                Mode = EditSymbol;
                XDefineCursor( display, XtWindow(draw_area), basic_cursor );
                XDefineCursor( display, XtWindow(draw_area), basic_cursor );
                }
            upd_mode_panel();
            }
        else if ( event->xbutton.button == Button3 )
            {
            if ( (event->xbutton.state & ShiftMask) &&      /*shift down*/
                 (Mode == EditSymbol) )
                {

                /*
                 *  implode into this symbol; this will set the current
                 *  element to the name in the symbol
                 */

                Symbol *sym;

                if ( (sym = (Symbol *)get_sym_map_entry(w)) == NULL )
                    user_ack("can't find element to implode");
                else
                    {
                    if ( sym->symbol_type == GOTO )
                            implode( (char *)sym->text, sym->Sym.ElemSym.comp_type );
                    }
                }
```

```
        else if ( Mode == EditSymbol )

        /*
         *  right button edit
         */

            edit_text( w );
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  set_origin()
*
* MODULE FUNCTION:
*
*   This routine retrieves the window's attributes and sets the global origin
*   vars
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  set_origin( w )

    Widget      w;

{
    XWindowAttributes   attribs;

    /*
     *  get attributes of this symbol.
     */

    if (! XGetWindowAttributes(display,XtWindow(w),&attribs) )
        error_handler( ERR_SYM_ATTRIBS, "set_origin" );

    /*
     *  set corner and center values for this window for
     *  future moves.
     */

    newx = attribs.x;
    newy = attribs.y;
    originx = attribs.width / 2;
    originy = attribs.height / 2;
}
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    cbr_utils.c
*
* FILE FUNCTION:
*
*   This file contains various routines which help the callbacks.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /home/project/3531/Docu/GCB.spec.doc
*
*
* FILE MODULES:
*
*   audit()                     - checks symbols; highlights in red incomplete ones
*   audit_line()                - highlights in red a line and symbols it connects.
*   audit_symbol()              - highlights in red a symbol and its lines.
*   build_lan_select_popup()    - builds the popup to input target language type.
*   cbr_audit()                 - responds when user presses 'audit' or 'clear audit'
*   cbr_audit_on()              - responds when user presses audit on/off btn
*   cbr_clear_popup()           - takes down the ask, help, or user_ack popup.
*   cbr_exit()                  - confirms, writes defaults, and exits GCB.
*   cbr_language()              - responds when user selects target language in popup.
*   cbr_language_menu()         - pops up select target language popup
*   cbr_pos_done()              - processes the element/comp purpose updating.
*   cbr_purpose()               - responds when user selects update element/comp purpose
*
*   cbr_set_sym_display()       - redraws all IF/SET syms with logical or expr text.
*   cbr_snap()                  - responds when user presses snap on/off button.
*   cbr_text_proc()             - processes the Create Position popup text fields
*   cbr_tgl_purpose()           - toggles between displaying comp and element purpose.
*   highlight_sym()             - changes back and fore colors of symbol, redraws it.
*   highlight_sym_lines()       - highlights symbol's lines and connected symbols
*   load_curr_dir()             - sets the parameter to current working directory.
*
*****************************************<----->*****************************************/

#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>

#include "gcb.h"
#include "cbr.h"
#include "widgets.h"
#include "constants.h"
#include "element_file.h"
#include "cbr_utils.h"
#include "lines.h"
#include "fonts.h"

extern  int PopupStat[],
            Red;

int     language;
int     Ask_Resp;
int     uil = 1;
int     RED_LINE = 0;
int     BLACK_LINE = 1;

void    clear_audit();

/*
 *  header string for the target language popup.
 */

char str_header[] = "Select the target language for the file which will be generated from
 the \nlogic diagram by the builder. Use the 'Build' option in the 'Element' menu \nto ge
nerate the target file.";
```

```
/*************************************<----->*************************************
*
* MODULE NAME:  audit()
*
* MODULE FUNCTION:
*
*   This routine checks each symbol for completeness, then highlights in red the
*   incomplete symbols and their lines.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*************************************<----->*************************************/

int  audit( type )

    int type;

{
    int i, ok = 1;

    /*
     *  redraw all the symbols so we can run audit twice in a row and
     *  highlight only the offending symbols each time.
     */

    for ( i=0; i<MAX_SYMBOLS; i++ )
            if ( Symbol_Map[i].symbol_type != NONE )
                if ( Zoomed )
                    redraw_sym_num( i, teeny_font );
                else redraw_sym_num( i, Symbol_Map[i].font );


    /*
     *  run through all the symbols, check each for completeness, highlight
     *  incomplete ones.
     */

    for ( i=0; i<MAX_SYMBOLS; i++ )
        switch ( i = complete(i, type) )
            {
            case -1:

                /*
                 *  We have run thru the whole Symbol_Map;
                 *  either the element is complete or we have highlighted every
                 *  incomplete symbol. ok is 1 if no symbol is incomplete.
                 */

                return( ok );
            case NO_BEGIN:
                user_ack("no BEGIN symbol!");
                return( 0 );
            case NO_END:
                user_ack("no END symbol!");
                return( 0 );
            default:

                /*
                 *  Save current background of the symbol,
                 *  highlight the offending Symbol_Map[i].mycanvas
                 *  and restore previous background for future symbol placement.
                 */

                highlight_sym( &Symbol_Map[i] );
                ok = 0;
                break;
            }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  audit_line()
*
* MODULE FUNCTION:
*
*   This routine highlights in red a line and the symbols it connects.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  audit_line()

{
    Line            *templine;
    Symbol          *from, *to;

    /*
     *  find to and from symbols.
     */

    templine    = (Line *)Cell_Map[LDlineY][LDlineX].cell_entry.lines->line;
    from        = (Symbol *)templine->from;
    to          = (Symbol *)templine->to;

    /*
     *  Redraw it in red...
     */

    draw_whole_line( templine, RED_LINE );

    /*
     *  Save current background of the symbols,
     *  highlight the connecting from-> and to->mycanvas
     *  and restore previous background for future symbol draw.
     */

    highlight_sym( from );
    highlight_sym( to );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  audit_symbol()
*
* MODULE FUNCTION:
*
*   This routine highlights in red a symbol and its lines.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  audit_symbol( w )

    Widget  w;

{
    int         sym;

    for ( sym=0; sym<MAX_SYMBOLS; sym++ )
        if ( Symbol_Map[sym].mycanvas == w )
            {

            /*
             *  highlight this symbol
             */

            highlight_sym( &Symbol_Map[sym] );

            /*
             *  highlight this symbol's lines;
             */

            highlight_sym_lines( &Symbol_Map[sym], TRUE );
            return;
            }
    user_ack("Fatal Error: audit_symbol: can't find symbol");
    exit( ERR );
}
```

```
/*****************************************<-----*****************************************
*
* MODULE NAME:  build_lan_select_popup()
*
* MODULE FUNCTION:
*
*   This routine builds the popup to input target language type.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<-----*****************************************/

Widget  build_lan_select_popup( parent )

    Widget parent;

{
    Arg             args[2];

    dlg_tl_select = cr_popup( NULLS, parent, "Select Target Language" );

    FormW = cr_form( NULLS, dlg_tl_select, NULL, NULL );
    set_attribs( FORM, FormW, 500, 300, XmRESIZE_NONE );

    lbl_tl_header = cr_label( NULLS, FormW,str_header, 0, 1, 30, 5, 95);
    XtSetArg( args[0], XmNalignment, XmALIGNMENT_BEGINNING );
    XtSetValues( lbl_tl_header, args, 1 );

    cr_label( NULLS, FormW, "Target Language:",0,42,46,14,IGNORE );

    rb_tl_select = cr_radio_box( NULLS, FormW, XmVERTICAL );
    set_position( rb_tl_select, 41, IGNORE, 60, IGNORE );

    tgl_language_c    = cr_toggle( NULLS, rb_tl_select, "C",    NULL, 0,0 );
    tgl_language_moal = cr_toggle( NULLS, rb_tl_select, "MOAL", NULL, 0,0 );
    tgl_language_uil  = cr_toggle( NULLS, rb_tl_select, "UIL",  NULL, 0,0 );

    XtAddCallback( tgl_language_c,    XmNarmCallback, cbr_language, 1 );
    XtAddCallback( tgl_language_moal, XmNarmCallback, cbr_language, 2 );
    XtAddCallback( tgl_language_uil,  XmNarmCallback, cbr_language, 3 );

    cr_separator( NULLS, FormW, 80, IGNORE, 1, 99 );

    DoneW   = cr_command( NULLS, FormW, "Apply",  cbr_language, 4 );
    ResetW  = cr_command( NULLS, FormW, "Reset",  cbr_language, 5 );
    CancelW = cr_command( NULLS, FormW, "Cancel", cbr_language, 6 );
    HelpW   = cr_command( NULLS, FormW, "Help",   cbr_help,     TARGET_LANG );

    set_position( CancelW,  88, IGNORE,  3, IGNORE );
    set_position( DoneW,    88, IGNORE, 29, IGNORE );
    set_position( ResetW,   88, IGNORE, 55, IGNORE );
    set_position( HelpW,    88, IGNORE, 80, IGNORE );
}
```

```
/*****************************************<-----*****************************************
*
* MODULE NAME:  cbr_audit()
*
* MODULE FUNCTION:
*
*   This routine responds when user presses 'audit' or 'clear audit' btn
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<-----*****************************************/

XtCallbackProc  cbr_audit( widget, closure, calldata )

    Widget      widget;
    int         closure;
    caddr_t     calldata;

{
    /*
     *  must be in editsymbol mode to audit
     */

    if ( Mode != EditSymbol )
        return;

    if ( closure == 4 )
        clear_audit();
    else

        /*
         *  audit for just connectivity, just expressions, or both.
         */

        audit( closure );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_audit_on()
*
* MODULE FUNCTION:
*
*   This routine responds when user presses audit on/off btn.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_audit_on( widget, closure, calldata )

    Widget      widget;
    int         closure;
    caddr_t     calldata;

{
    /*
     *  set global variable on/off.
     */

    if ( closure == 1 )
        Audit = 1;
    else Audit = 0;
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_clear_popup()
*
* MODULE FUNCTION:
*
*   This routine takes down the ask, help, or user_ack popup.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_clear_popup( widget, closure, calldata )

    Widget  widget;
    caddr_t closure,
            calldata;

{
     /*
      *  Determine which button the user pushed, act accordingly.
      */

    switch ( (int) closure )
        {
        case ASK_NO     : Ask_Resp = FALSE;
                          XtUnmanageChild( dlg_ask );
                          break;
        case ASK_YES    : Ask_Resp = TRUE;
                          XtUnmanageChild( dlg_ask );
                          break;
        case HELP_CANCEL: XtUnmanageChild( dlg_help );
                          break;
        case USER_ACK   : XtUnmanageChild( dlg_ack );
                          break;
        }
        PopupStat[ ActivePopup ] = 1;
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_exit()
*
* MODULE FUNCTION:
*
*   This routine confirms exit, writes defaults, and exits GCB.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.01 - 08/01/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_exit( widget, closure, calldata )

    Widget  widget;
    caddr_t closure,
            calldata;

{
    /*
     *  remind user if save is needed.
     */

    if ( SaveNeeded && ValidElement )
        if ( ask("Do you want to save your changes to the current Element file?") )
            save_element_file();

    if ( ask("Are you sure you want to exit the GCB?") )
        {
        write_defaults();
        exit( OK );
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_language()
*
* MODULE FUNCTION:
*
*   This routine responds when user selects target language in popup.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_language( widget, closure, calldata )

    Widget  widget;
    int     closure,
            calldata;

{
    if ( closure == 6 )

        /*
         *  cancel language select popup.
         */

        XtUnmanageChild( dlg_tl_select );

    else if ( closure == 4 )
        {

        /*
         *  user has selected language; set global var and popdown.
         */

        TargetLanguage = language;
        XtUnmanageChild( dlg_tl_select );
        }

    else if ( closure == 5 )
        {
        switch ( TargetLanguage )
            {
            case C      :     arm_tgl( tgl_language_c );
                          disarm_tgl( tgl_language_moal );
                          disarm_tgl( tgl_language_uil );
                          break;
            case MOAL   :     arm_tgl( tgl_language_moal );
                          disarm_tgl( tgl_language_c );
                          disarm_tgl( tgl_language_uil );
                          break;
            case UIL    :     arm_tgl( tgl_language_uil );
                          disarm_tgl( tgl_language_c );
                          disarm_tgl( tgl_language_moal );
                          break;
            default     :   elog(1,"cbr_language() - bad TargetLanguage type");
            }
        }
    else
        {
```

```
    if ( closure == 3 )

        /*
         *  this is a UIL toggle button release, show user_ack.
         */

        user_ack("Sorry, but UIL is currently not supported");

    /*
     *  set current language.
     */

    language = (int)closure;
    }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_language_menu()
*
* MODULE FUNCTION:
*
*   This routine sets the proper toggles and pops up the select target language popup.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_language_menu( widget, closure, calldata )

    Widget  widget;
    caddr_t closure,
            calldata;

{
    if ( Mode != EditSymbol )
        return;

    /*
     *  set toggle to current language
     */

    switch ( TargetLanguage )
        {
        case 0      :
        case C      :     arm_tgl( tgl_language_c );
                      disarm_tgl( tgl_language_moal );
                      disarm_tgl( tgl_language_uil );
                      break;
        case MOAL   :     arm_tgl( tgl_language_moal );
                      disarm_tgl( tgl_language_c );
                      disarm_tgl( tgl_language_uil );
                      break;
        case UIL    :     arm_tgl( tgl_language_uil );
                      disarm_tgl( tgl_language_c );
                      disarm_tgl( tgl_language_moal );
                      break;
        default     :   elog(1,"cbr_language() - bad TargetLanguage type");
        }

    XtManageChild( dlg_tl_select );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_pos_done()
*
* MODULE FUNCTION:
*
*   This routine processes the element file and comp file purpose updating.  This
*   routine is called when the user clicks either DONE or CANCEL in the purpose
*   update popup.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.01 - 08/01/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_pos_done( w, client_data, calldata )

    Widget  w;
    int     client_data;
    caddr_t calldata;

{
    int     length;

    if ( client_data == CANCEL )
        {
        XtUnmanageChild( dlg_purpose );
        return;
        }

    /*
     *  Clear the READ-ONLY attribute and then clear the widget.  Install
     *  the new purpose text into the widget.
     */

    XmTextSetEditable( txt_purpose, True );
    length = strlen( XmTextGetString(scr_purpose) );

    /*
     *  Determine which type of purpose we are updating: Comp or Element.  Then
     *  place the new text in the proper string based on the type Comp or Element.
     */

    if ( XmToggleButtonGetState(tgl_element) )
        {
        strcpy( ElementPurpose, XmTextGetString(scr_purpose) );
        ElementPurpose[length] = '\0';
        XmTextSetString( txt_purpose, ElementPurpose );

        /*
         *  Mark the Element file as updated and in need of being saved to disk.
         */

        SaveNeeded = True;
        }
    else
        {
        strcpy( CompPurpose, XmTextGetString(scr_purpose) );
        CompPurpose[length] = '\0';
        XmTextSetString( txt_purpose, CompPurpose );

        /*
         *  Update the comp file with the new purpose text.
         */

        update_comp_file( GCompFile, CompPurpose, RootElement, NULL, NULL );
        }

    /*
     *  Reset the READ-ONLY attribute and then take down the frame.
     */

    XmTextSetEditable( txt_purpose, False );
    XtUnmanageChild( dlg_purpose );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_purpose()
*
* MODULE FUNCTION:
*
*   This routine responds when the user selects update element/comp purpose.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_purpose( widget, closure, calldata )

    Widget  widget;
    caddr_t closure,
            calldata;

{
    if ( Mode != EditSymbol )
        return;

    /*
     *  Depending on which type of purpose the user wants to edit, load the
     *  popup's textsw with the correct purpose text from the global vars.
     *  Then set the main screen's purpose type button to the correct type
     *  and then show the popup.
     */

    if ( (int) closure == 0 )
        {
        XmTextSetString( scr_purpose, CompPurpose );
           arm_tgl( tgl_comp );
        disarm_tgl( tgl_element );
        }
    else
        {
        XmTextSetString( scr_purpose, ElementPurpose );
           arm_tgl( tgl_element );
        disarm_tgl( tgl_comp );
        }

    XtManageChild( dlg_purpose );

    /*
     *  Update the position panel.  This will update the text in the main
     *  screen's purpose textsw so the text in the main screen will match
     *  that of the popup.
     */

    upd_pos_panel( NO_CHANGE );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_text_proc()
*
* MODULE FUNCTION:
*
*   This routine updates the text fields in the Create Position popup. The Create
*   Position popup displays to the user the directory which will be created, making
*   sure the directory extension is correct.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_text_proc ( widget, closure, calldata )

    Widget                  widget;
    int                     closure;
    XmAnyCallbackStruct     *calldata;

{
    Arg     .     args[1];
    XmString    tcs;
    int         currpos;

    /*
     *  Process the create Position text fields and push buttons.
     */

    if ( (closure == 1) || (closure == 0) )
        {

        /*
         *  Get the specified path, remove the position directory
         *  name if one exists at the end of the path.
         */

        strcpy( newPath,  XmTextGetString(txt_cre_pos_ppath) );
        justPath( newPath );
        XmTextSetString(txt_cre_pos_ppath, newPath);

        /*
         *  Get the name specified by the user.  Remove the position
         *  directory extension if one exists on the user specified
         *  name.
         */

        strcpy( newName, XmTextGetString(txt_cre_pos_ne) );
        justName( newName );
        XmTextSetString(txt_cre_pos_ne, newName);

        strcat( newPath, newName );
        strcat( newPath, ".POS" );

        /*
         *  Update the label which displays to the user what the new directory
         *  name is that will be created.
         */
```

```
        tcs = XmStringCreate( newPath, XmSTRING_DEFAULT_CHARSET );
        XtSetArg( args[0], XmNlabelString, tcs );
        XtSetValues( txt_cre_pos_nd, args, 1 );
        XmStringFree( tcs );
        }
    else if ( closure == 2 )
        {
        XmRemoveTabGroup( dlg_cre_comp );

        /*DBEUG*/
        elog(3,"textproc: activate\n");

        XSetInputFocus( display, XtWindow(XtParent(txt_cre_comp_re_name)),
                RevertToPointerRoot, CurrentTime );
        XtSetKeyboardFocus( dlg_cre_comp, XtParent(txt_cre_comp_re_name) );
        }
    else if ( closure == 3 )
        XSetInputFocus( display, XtWindow(scr_cre_comp), RevertToPointerRoot,
                        CurrentTime );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_tgl_purpose()
*
* MODULE FUNCTION:
*
*   This routine toggles between displaying the comp and the element purpose.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_tgl_purpose( widget, closure, calldata )

    Widget  widget;
    caddr_t closure,
            calldata;

{
    if ( (int)closure == 2 )

    /*
     *  display comp purpose
     */

        XmTextSetString( txt_purpose, CompPurpose );
    else if ( (int)closure == 3 )
        XmTextSetString( txt_purpose, ElementPurpose );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_set_sym_display()
*
* MODULE FUNCTION:
*
*   This routine redraws all IF/SET syms with logical or expr text.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_set_sym_display( widget, closure, calldata )

    Widget  widget;
    caddr_t closure,
            calldata;

{
    /*
     *  set global var so draw routine knows which text to display.
     */

    if ( (int)closure == 1 )
        LogOrCompText = 0;
    else
        LogOrCompText = 1;

    /*
     *  redraw all if/set symbols with proper text displayed.
     */

    redraw_sym_type( IF );
    redraw_sym_type( SET );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_snap()
*
* MODULE FUNCTION:
*
*   This routine responds when the user presses the snap on/off button.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_snap( widget, closure, calldata )

    Widget  widget;
    caddr_t closure,
            calldata;
{
    if ( (int)closure == 1 )
        Snap = 1;
    else Snap = 0;
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  clear_audit()
*
* MODULE FUNCTION:
*
*   This routine redraws all symbols in their proper color after an audit.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  clear_audit()

{
    int         i;
    LineList    *linelist;
    Line        *lineptr;

    for ( i=0; i<MAX_SYMBOLS; i++ )
        if ( Symbol_Map[i].symbol_type != NONE )
            {

            /*
             *  redraw this symbol with proper font
             */

            if ( Zoomed )
                redraw_sym_num( i, teeny_font );
            else redraw_sym_num( i, Symbol_Map[i].font );

            /*
             *  redraw lines entering symbol
             */

            linelist = Symbol_Map[i].from;
            while ( linelist )
                {
                if ( (lineptr = linelist->line) != NULL )
                  ! {
                    /*
                     *  redraw next "from" line in black
                     */

                    draw_whole_line( lineptr, BLACK_LINE );
                    }
                linelist = (LineList *)linelist->next;
                }
            }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  highlight_sym()
*
* MODULE FUNCTION:
*
*   This routine changes the back and foreground colors of a symbol, then
*   redraws it.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  highlight_sym( sym )

    Symbol  *sym;
{
    unsigned long   temp_back, temp_fore;

    /*
     *  store current colors.
     */

    temp_back   = get_my_background( sym->symbol_type );
    temp_fore   = get_my_foreground( sym->symbol_type );

    /*
     *  set new colors
     */

    if (Color)
        {
        set_my_background(sym->symbol_type, Red );
        set_my_foreground(sym->symbol_type, BlackPixel(display,DefaultScreen(display)));
        }
     else
        {
        set_my_background(sym->symbol_type, WhitePixel(display,DefaultScreen(display)));
        set_my_foreground(sym->symbol_type, BlackPixel(display,DefaultScreen(display)));
        }

    if ( Zoomed )
        redraw_sym_num( compute_label_index(sym), teeny_font );
    else
        redraw_sym_num( compute_label_index(sym), sym->font );

    /*
     *  reset current colors.
     */

    set_my_background( sym->symbol_type, (unsigned long)temp_back );
    set_my_foreground( sym->symbol_type, (unsigned long)temp_fore );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  highlight_sym_lines()
*
* MODULE FUNCTION:
*
*   This routine highlights a symbol's lines; optionally, the symbols
*   connected to it.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  highlight_sym_lines( sym, endsymbols )

    Symbol  *sym;
    int     endsymbols;

{
    Line        *lineptr;
    LineList    *linelist;

    /*
     *  highlight the lines (and symbols) connected to this symbol.
     */

    linelist = (LineList *)sym->from;
    while ( linelist )
        {
        lineptr = (Line *)linelist->line;
        if ( endsymbols )
            highlight_sym( lineptr->from );

        /*
         *  redraw line in red.
         */

        draw_whole_line( lineptr, RED_LINE );
        linelist = (LineList *)linelist->next;
        }
    /*
     *  highlight the lines (and symbols) connected from this symbol.
     */

    if ( sym->symbol_type == IF )
        {
        if ( lineptr = (Line *)sym->Sym.IfSym.true_line )
            {
            if ( endsymbols )
                highlight_sym( lineptr->to );
            /*
             *  redraw line in red.
             */

            draw_whole_line( lineptr, RED_LINE );
            }
        if ( lineptr = (Line *)sym->Sym.IfSym.false_line )
            {
            if ( endsymbols )
                highlight_sym( lineptr->to );
            /*
             *  redraw line in red.
             */

            draw_whole_line( lineptr, RED_LINE );
            }
        }
    else
        {
        if ( lineptr = (Line *)sym->next )
            {
            if ( endsymbols )
                highlight_sym( lineptr->to );

            /*
             *  redraw line in red.
             */

            draw_whole_line( lineptr, RED_LINE );
            }
        }
}
```

```
/*************************************<----->*************************************
*
* MODULE NAME:  load_curr_dir()
*
* MODULE FUNCTION:
*
*   This routine sets the parameter to the current working directory.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*************************************<----->*************************************/

void  load_curr_dir( cwd )

    char *cwd;

{
    if (! getwd(cwd) )
        {
        user_ack("Error: couldn't get working directory");
        elog(1,"load_curr_dir() - couldn't getcwd()");
        return;
        }
}
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    cbr_utils.h
*
*
* FILE FUNCTION:
*
*   Global variable declarations for the cbr_utils.c routines.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /home/project/3531/Docu/GCB.spec.doc
*
*
* FILE MODULES:
*
*   N/A
*
*****************************************<----->*****************************************/

extern char     newName[MAX_NAME],      /*  name of Position directory to create    */
                newPath[MAX_PATH];      /*  path to Position directory during create*/
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    cbr_var_input.c
*
* FILE FUNCTION:
*
*   This file contains the routines which accept variable declarations from
*   the user.
*
*
* FILE MODULES:
*
*   add_var_to_sym_table() - adds variable to symbol table using matrix, type toggles.
*   cbr_num_input_done  () - handles events when user enters numbers in popup.
*   cbr_str_input       () - processes the button presses in the String popup.
*   cbr_var_choose      () - sets fields when user selects name in var_input popup
*   cbr_var_input_done  () - responds when user enters var names, numbers, or text
*   cbr_var_type        () - records variable type chosen in var declaration popup.
*   count_quotes        () - counts quotes in string; ensures they are properly placed
*   destroy_global_varlist() -  deallocates space needed for global varlist.
*   itoa                () - returns the ascii equivalent of the number parameter.
*   load_variable_list  () - loads list from which to select variables
*   lookup_global_varlist() - forms list of global variables for the current element.
*   right_type()            - determines if name parameter fits type parameter.
*   valid_name          () - determines if name has proper type chars, # of brackets.
*   valid_num           () - determines if the number is well_formed.
*
*****************************************<----->*****************************************/

#include <stdio.h>

#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include <Xm/Text.h>

#include "gcb.h"
#include "widgets.h"
#include "symbol.h"
#include "gcb_parse.h"
#include "lines.h"
#include "next_inputs.h"
#include "menu.h"
#include "cbr_var_input.h"
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  add_var_to_sym_table()
*
* MODULE FUNCTION:
*
*   This routine adds a new variable to the symbol table using the matrix and
*   type toggles.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  add_var_to_sym_table()

{
    int         attributes = 0, entry_val, matrix = 0;
    Arg         args[1];
    short       num_dims = 0, rows = 0, cols = 0;
    short       dim3 = 0, dim4 = 0;

    /*
     *  set attribute of new variable; default is integer.
     */

    attributes |= VARIABLE;
    if ( !rb_sym_type )
        rb_sym_type = INTEGER;
    attributes |= rb_sym_type;

    if ( XmToggleButtonGetState(tgl_is_mat) )
        {
        attributes |= MATRIX;

        /*DEBUG*/
        elog(3,"its a matrix! %d by %d, attribs = %o",
                rows = (short)atoi(XmTextGetString(txt_matdim_x)),
                cols = (short)atoi(XmTextGetString(txt_matdim_y)),
                dim3 = (short)atoi(XmTextGetString(txt_matdim_3)),
                dim4 = (short)atoi(XmTextGetString(txt_matdim_4)),
                attributes  );

        /*
         *  default # of dimensions is 2.
         */

        num_dims = 2;
        if ( (short)atoi(XmTextGetString(txt_matdim_3)) )
            num_dims = 3;
        if ( (short)atoi(XmTextGetString(txt_matdim_4)) )
            num_dims = 4;

        if ( (cols < 1) || (rows < 1) ||
           ((num_dims == 3) && (dim3 < 1)) ||
           ((num_dims == 4) && (dim4 < 1)) )
           {
           user_ack("Matrix dimensions must be greater than zero");
           return( 0 );
           }
```

```
        }

    /*DEBUG*/
    elog(3,"add var: rb sym type = %i", rb_sym_type);

    /*
     *  Add the variable to the symbol table, make its parent in the symbol table
     *  dependent on the variable scope.
     */

    if ( right_type(LOCAL_VAR,variable) )
        entry_val = add_symbol_entry( ElementFile, variable, attributes, num_dims,
            rows, cols, dim3, dim4 );
    else
        entry_val = add_symbol_entry( NULL, variable, attributes, num_dims,
            rows, cols, dim3,dim4 );

    if ( entry_val )
        error_handler( ERR_ADD_SYMBOL, "add_var_to_sym_table" );

    /*DEBUG*/
    elog(3,"added %s %s %i to sym table, return val %d", ElementFile, variable,
            attributes, entry_val);

    return( 1 );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_num_input_done()
*
* MODULE FUNCTION:
*
*   This routine handles events when the user is finished entering numbers
*   in a popup window.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_num_input_done( w, closure, call_data )

    Widget      w;
    int         closure;
    caddr_t     *call_data;

{
    int popup_type;
    Arg args[1];

    if ( closure == CBR_COMP_TYPE_CANCEL )
        {

        /*
         *  reset text field and popdown.
         */

        XmTextSetString( scr_var_input, NULLS );
        XtUnmanageChild( dlg_var_input );
        }
    else if ( closure == CBR_COMP_TYPE_DONE )
        {

        /*
         *  the type of the popup is hidden in the userData resource.
         */

        XtSetArg( args[0], XmNuserData, &popup_type );
        XtGetValues( dlg_var_input, args, 1 );

        /*DEBUG*/
        elog(3,"str input done: popup type is %i", popup_type);

        if ( popup_type == NUMBER )

        /*
         *  is number well-formed?
         */

            if (! valid_num( XmTextGetString(scr_var_input) ) )
                {
                user_ack("Your number contained an illegal char");
                return;
                }

        XmTextInsert( scr_expr, XmTextGetInsertionPosition(scr_expr),
```

```
        XmTextGetString(scr_var_input) );
    XmTextSetString( scr_var_input, NULLS );
    XtUnmanageChild( dlg_var_input );

    /*
     *  parse the expression obtained so far.
     */

    do_parse();
    }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_str_input_done()
*
* MODULE FUNCTION:
*
*   This is the callback routine which is called when the user is adding a String
*   to a logical expression.  This routine processes the button presses in the
*   String popup.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_str_input_done( w, closure, call_data )

    Widget      w;
    int         closure;
    caddr_t     *call_data;

{
    char *str;
    int  rc;


    if ( closure == CANCEL )
        {
        XtUnmanageChild( dlg_str_input );
        return;
        }

    /*
     *  User chose DONE, insert the string into the expression.  If the user didn't
     *  put in any quotes, add them.  If the user put in the quotes correctly, don't
     *  mess with the original string, just add it to the expression.  If the user
     *  did something wrong, scold them and don't add the string to the expression.
     */

    str = XmTextGetString( txt_str_input );
    rc = count_quotes( str );
    if ( rc == 0 )
        {
        XmTextInsert( scr_expr, XmTextGetInsertionPosition(scr_expr), "\"" );
        XmTextInsert( scr_expr, XmTextGetInsertionPosition(scr_expr), str  );
        XmTextInsert( scr_expr, XmTextGetInsertionPosition(scr_expr), "\"" );
        }
    else if ( rc == 2 )
        XmTextInsert( scr_expr, XmTextGetInsertionPosition(scr_expr), str );
    else if ( rc == ERR )
        {
        user_ack("Your string is improperly formed, please remove quotes -> \"");
        return;
        }

    XtUnmanageChild( dlg_str_input );
    do_parse();
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_var_input_done()
*
* MODULE FUNCTION:
*
*   This routine handles events when the user is finished entering
*   variable names, numbers, or text in a popup  window.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_var_input_done( w, closure, call_data )

    Widget      w;
    int         closure;
    caddr_t     *call_data;

{
    Arg     args[1];
    char    newStr[ MAX_TEXT ];
    int     sym_type,
            popup_type;

    /*
     *  See if the user wants to CANCEL selecting a variable.
     */

    if ( closure == CANCEL )
        {
        XmTextSetString( txt_var_name, NULLS );
        XtUnmanageChild( dlg_logic_type );
        return;
        }

    /*
     *  Can't have STRING MATRIX or STRING VECTOR.
     */

    if ( (XmToggleButtonGetState(tgl_is_mat)) && (rb_sym_type == CHAR) )
        {
        user_ack("The GCB currently does not support STRING MATRICES");
        return;
        }

    /*
     *  User selected DONE, get the text from the selection box widget.  Remove the
     *  matrix element specification from the "variable" because we are going to
     *  use him to do symbol table ops and we don't want the matrix element brackets
     *  in the string.  The "newStr" variable will keep the matrix element brackets.
     */

    strcpy( variable, XmTextGetString(txt_var_name) );
    strcpy( newStr,   variable );
    just_mat_name( variable );

    XtSetArg( args[0], XmNuserData, &popup_type );
    XtGetValues( dlg_logic_type, args, 1);
```

```
/*
 *  Get the text from the matrix element text fields.  Append them to the variable
 *  name.  First make sure the user didn't type in the brackets with the name.  If
 *  they did, too bad, any values in the Element text fields will replace what the
 *  user typed in.
 */

if ( strlen(XmTextGetString(txt_matelm_x)) )
    {
    just_mat_name( newStr );
    strcat( newStr, "[" );
    strcat( newStr, XmTextGetString(txt_matelm_x) );

    if ((popup_type == LOCAL_VAR) || (popup_type == GLOBAL_VAR))
        {
        strcat( newStr, "][" );
        strcat( newStr, XmTextGetString(txt_matelm_y) );
        strcat( newStr, "]" );

        if( atoi(XmTextGetString(txt_matelm_3)) )
            {
            strcat( newStr, "[" );
            strcat( newStr, XmTextGetString(txt_matelm_3) );
            strcat( newStr, "]" );
            }
        if( atoi(XmTextGetString(txt_matelm_4)) )
            {
            strcat( newStr, "[" );
            strcat( newStr, XmTextGetString(txt_matelm_4) );
            strcat( newStr, "]" );
            }
        }
    else
        strcat( newStr, "]" );
    }
else

    /*
     *  The user may have typed in a matrix element specifier, check the name
     *  user entered including the brackets.
     */

    if ( valid_name(newStr,popup_type) )
        {
        user_ack("The variable name contains an invalid character or prefix");
        return;
        }

/*
 *  Make sure the identifier/variable name is properly formed.
 */

if ( valid_name(variable,popup_type) )
    {
    user_ack("The variable name contains an illegal character or invalid prefix");
    return;
    }

/*
 *  See if the identifier/variable name is in the symbol table.
 */

if (! (struct symbol_entry *)lookup_symbol( (popup_type == LOCAL_VAR) ?
    ElementFile : NULL, variable) )
    {

    /*
     *  This identifier is not in the symbol table, determine which type of
     *  symbol we are building.
     */

    XtSetArg( args[0], XmNuserData, &sym_type );
    XtGetValues( current_symbol, args, 1 );

    /*
     *  If we have a local variable which is on the RHS and is not in the
     *  symbol table, let the user know they must set a local variable before
     *  using it.  We let the user use globals before being set because they
     *  may be designing "TopUp".
     */

    if ( (popup_type == LOCAL_VAR) && ((sym_type == IF) || (WhereAmI == RHS)) )
        {
        user_ack("Must SET/initialize variable before using in an expression");
        return;
        }

    /*
     *  This is the first reference to an identifier and it is not a local
     *  variable, add the identifier to the symbol table.
     */

    else if (! add_var_to_sym_table() )
        {
        XmTextSetString( txt_matelm_x, NULLS );
        XmTextSetString( txt_matelm_y, NULLS );
        XmTextSetString( txt_matelm_3, NULLS );
        XmTextSetString( txt_matelm_4, NULLS );
        XmTextSetString( txt_var_name, NULLS );
        XtUnmanageChild( dlg_logic_type );
        return;
        }
    }

/*
 *  Take down the Select variable popup.
 */

XmTextSetString( txt_matelm_x, NULLS );
XmTextSetString( txt_matelm_y, NULLS );
XmTextSetString( txt_matelm_3, NULLS );
XmTextSetString( txt_matelm_4, NULLS );
XmTextSetString( txt_var_name, NULLS );
XtUnmanageChild( dlg_logic_type );

/*
 *  Reset the rb_sym_type variable, which tells the most
 *  recently selected variable type toggle.
 */

rb_sym_type = INTEGER;

/*
 *  Insert the identifier/variable name into the expression.
 */

XmTextInsert( scr_expr, XmTextGetInsertionPosition(scr_expr), newStr );
```

```
    /*
     *  parse the expression obtained so far.
     */

    do_parse();
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_var_choose()
*
* MODULE FUNCTION:
*
*   This routine sets the text fields according to the variable name selected by
*   the user in the var_input popup.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                                  Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_var_choose( w, closure, call_data )

    Widget                          w;
    int                             closure;
    XmListCallbackStruct            *call_data;

{
    char    *str, tempName[MAX_NAME], r[3], c[3], d3[3], d4[3];
    Arg     args[1];
    int     intdim3, intdim4, popup_type, type, introws, intcols, itoa();
    short   dims, dim3, dim4, rows, cols;

    XmStringGetLtoR( call_data->item, XmSTRING_DEFAULT_CHARSET, &str );
    sscanf( str, "%s", tempName );
    XmTextSetString( txt_var_name, tempName );

    /*
     *  lookup symbol; get symbol type and set rb button; set matrix dims
     *  if appropriate.
     */

    XtSetArg( args[0], XmNuserData, &popup_type );
    XtGetValues( dlg_logic_type, args, 1 );

    if ( (type = return_symbol_attributes( (popup_type != LOCAL_VAR) ?
        NULL : ElementFile, tempName)) == ERR )
            {

            /*
             *  Panic: the variable was loaded into the list from the symbol table
             *  but is no longer there.
             */

            user_ack("variable in list but not in symbol table");
            exit(ERR);
            }

    /*
     *  set toggle values according to var type.
     */

    switch( rb_sym_type )
        {
        case INTEGER:   disarm_tgl( tgl_var_type_int );
            break;
        case FLOAT:     disarm_tgl( tgl_var_type_real );
```

```
        break;
    case CHAR:      disarm_tgl( tgl_var_type_string );
        break;
    case DOUBLE:    disarm_tgl( tgl_var_type_double );
        break;
    case UNSIGNED:  disarm_tgl( tgl_var_type_unsigned );
        break;
    case SHORT:     disarm_tgl( tgl_var_type_short );
        break;
    }
disarm_tgl( tgl_var_type_int );

/*
 *  arm proper toggle and set selected type.
 */

if ( type & INTEGER )
    {
    arm_tgl( tgl_var_type_int );
    rb_sym_type = INTEGER;
    }
else if ( type & FLOAT )
    {
    arm_tgl( tgl_var_type_real );
    rb_sym_type = FLOAT;
    }
else if ( type & CHAR )
    {
    arm_tgl( tgl_var_type_string );
    rb_sym_type = CHAR; }
else if ( type & DOUBLE )
    {
    arm_tgl( tgl_var_type_double );
    rb_sym_type = DOUBLE;
    }
else if ( type & UNSIGNED )
    {
    arm_tgl( tgl_var_type_unsigned );
    rb_sym_type = UNSIGNED;
    }
else if ( type & SHORT )
    {
    arm_tgl( tgl_var_type_short );
    rb_sym_type = SHORT;
    }

if ( type & MATRIX )
    {

    /*
     *  set dimension text fields
     */

    arm_tgl( tgl_is_mat );
    disarm_tgl( tgl_isnt_mat );

    cbr_mat_tgls( NULL, MAT, NULL );

    if ( (type = return_matrix_attributes( (popup_type != LOCAL_VAR) ?
        NULL : ElementFile, tempName, &dims, &rows, &cols, &dim3, &dim4)) == ERR )
            {
            /*DEBUG*/
            elog(1,"choose: retur matrix attribs failed for %s", tempName);
            user_ack("can't find matrix attributes in cbr_var_choose");
            exit( ERR );
            }

        /*
         *  convert short to int.
         */

        introws = rows;
        intcols = cols;

        /*
         *  convert int to char string and set dimension text fields.
         */

        itoa( introws, r );
        itoa( intcols, c );

        /*
         *  if dims > 4 or dims > 3, convert 3 and 4 dimensions to strings.
         *  else clear 3rd and 4th dimension fields.
         */

        if ( dims == 2 )
            {
            XmTextSetString( txt_matdim_3, NULLS );
            XmTextSetString( txt_matdim_4, NULLS );
            }
        if ( dims > 2 )
            {
            intdim3 = dim3;
            itoa( intdim3, d3 );
            }
        if ( dims > 3 )
            {
            intdim4 = dim4;
            itoa( intdim4, d4 );
            }

        XmTextSetString( txt_matdim_x, r );
        XmTextSetString( txt_matdim_y, c );

        if ( dims > 2 )
            XmTextSetString( txt_matdim_3, d3 );
        if ( dims > 3 )
            XmTextSetString( txt_matdim_4, d4 );
        }

    /*
     *  The selected variable is not a matrix variable, reset the toggle buttons
     *  and clear the matrix dimension text fields.
     */

    else
        {
           arm_tgl( tgl_isnt_mat );
        disarm_tgl( tgl_is_mat );

        cbr_mat_tgls( NULL, SCAL, NULL );
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_var_type()
*
* MODULE FUNCTION:
*
*   This routine  records the variable type chosen by the user in the variable
*   declaration popup.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_var_type( w, closure, call_data )

    Widget      w;
    int         closure;
    caddr_t     call_data;

{
    rb_sym_type = closure;
    /*DEBUG*/
    elog(3,"setting rb_sym_type to %i", rb_sym_type);
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  count_quotes()
*
* MODULE FUNCTION:
*
*   This routine counts the number of quotes in a string and makes sure the quotes
*   are in the proper place in the string.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  count_quotes( str )

    char *str;

{
    char *ptr;
    int  count = 0;

    /*
     *  Count the number of quotes in the string.
     */

    ptr = str;
    while ( *ptr != NULL )
        {
        if ( *ptr == '\"' )
            count++;
        ptr++;
        }

    /*
     *  If there are no quotes or if there are too many quotes, then return.
     */

    if ( count == 0 )
        return( count );
    if ((count > 2) || (count == 1))
        return( ERR );

    /*
     *  There are two quotes, make sure they are in the right spots.
     */

    if ((str[0] != '\"') || (str[strlen(str)-1] != '\"'))
        return( ERR );

    return( 2 );
}
```

```
/*****************************************<-----*****************************************
*
* MODULE NAME:  destroy_global_varlist()
*
* MODULE FUNCTION:
*
*   This routine deallocates space needed for global varlist.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<-----*****************************************/

int  destroy_global_varlist( list )

    struct symbol_entry *list;

{
    struct symbol_entry *temp;

    while ( list )
        {
        temp = list;
        list = (struct symbol_entry *)list->se_next;
        free( temp );
        }
}
```

```
/*****************************************<-----*****************************************
*
* MODULE NAME:  right_type()
*
* MODULE FUNCTION:
*
*   This routine  determines if its name parameter fits its type parameter.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<-----*****************************************/

int  right_type( type, name )

    int     type;
    char    *name;

{
    int vtype = LOCAL_VAR;

    /*
     *  Determine the type of the variable based on its name.
     */

    if ( (name[0] == 'G') && (name[1] == 'V') &&  (name[2] == '_') )
        vtype = GLOBAL;

    else if ( name[0] == 'W' )
        vtype = WSG;

    else if ( name[0] == 'V' )
        vtype = MSID;

    /*
     *  See if the data type of the variable matches what we are looking for.
     */

    if ( type == vtype )
        return( True );
    else
        return( False );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  itoa()
*
* MODULE FUNCTION:
*
*   This routine returns the ascii equivalent of the number parameter.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  itoa( n, s )

    int     n;
    char    *s;

{
    int i = 0, j, k, c;

    /*
     *  convert each digit starting from the 1's place to a char.
     */

    do
        {
        s[i] = (n % 10) + 48;
        n = n / 10;
        i++;
        }
    while ( n );

    /*
     *  reverse string
     */

    for ( j = 0, k = i - 1; j < k; j++, k-- )
        {
        c = s[j];
        s[j] = s[k];
        s[k] = c;
        }

    s[i] = '\0';
    return( i );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  just_mat_name()
*
* MODULE FUNCTION:
*
*   This routine removes the "[num][num]" from a matrix name.  The input string
*   is modified so only the basename portion of a matrix name remains.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  just_mat_name( name )

    char *name;

{
    char *ptr;

    if (! name )
        return;

    ptr = name;
    while ( (*ptr!='[') && (*ptr!=']') && (*ptr!=NULL) )
        ptr++;

    *ptr = NULL;
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  load_variable_list()
*
* MODULE FUNCTION:
*
*   This routine loads the list from which the user selects variables, according
*   to the button the user clicked to bring up the popup.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  load_variable_list( selbox, type )

    Widget  selbox;
    int     type;

{
    struct symbol_entry     *varlist, *lookup_global_varlist();
    Widget                  list;
    char                    selList[MAX_NAME][MAX_NAME];
    int                     i, j, global = 0;
    XmString                xmstr;

    list = (Widget) XmSelectionBoxGetChild( selbox, XmDIALOG_LIST );

    /*
     *  Call the appropriate function to return a variable list.
     */

    switch ( type )
        {
        case LOCAL_VAR :

            varlist = (struct symbol_entry *) lookup_local_varlist( ElementFile );
            if ( varlist )
                elog(3,"load var list: found local var list for elem file %s",
                        ElementFile);
            else
                elog(3,"load var list: no find local var list for elem file %s",
                        ElementFile);
            break;

        case GLOBAL :
        case MSID :
        case WSG :

            varlist = (struct symbol_entry *) lookup_global_varlist();
            global = 1;
            break;
        }

    /*
     *  delete the current list entries.
     */

    XmListDeleteAllItems( list );

    /*
     *  go thru list, extract var name from each entry
     */

    i = 0;
    while ( varlist )
        {

        /*
         *  We only want variables of the right type for this type of list.
         */
        if ( right_type(type,varlist->se_symbol) )
            {
            /*DEBUG*/
            elog(3,"load_var_list:  loading var name %s", varlist->se_symbol);

            strcpy( selList[i], varlist->se_symbol );

            /*DEBUG*/
            elog(3,"load_var_list: symbol %s", varlist->se_symbol);

            /*
             *  Append some spaces to the variable name so we can then add
             *  the data type information.
             */

            for ( j=0; j < (20-strlen(varlist->se_symbol)); j++ )
                strcat( selList[i], " " );

            if ( varlist->se_type & INTEGER )
                strcat( selList[i], "int" );
            else if ( varlist->se_type & UNSIGNED )
                strcat( selList[i], "unsigned int" );
            else if ( varlist->se_type & FLOAT )
                strcat( selList[i], "float" );
            else if ( varlist->se_type & SHORT )
                strcat( selList[i], "short" );
            else if ( varlist->se_type & DOUBLE )
                strcat( selList[i], "double" );
            else if ( varlist->se_type & CHAR )
                strcat( selList[i], "string" );

            /*
             *  Check for a matrix, add "matrix" past the data type.
             */

            if ( varlist->se_type & MATRIX )
                strcat( selList[i], " matrix" );

            /*DEBUG*/
            elog(3,"%s", selList[i]);

            i++;
            }
        varlist = varlist->se_next;
        }

    /*
     *  Sort the list and install the list into the selection widget.
     */

    if ( i > 0 )
        {
        for ( j=0; j<i; j++ )
```

```
        {
        }
    qsort( selList, i, MAX_NAME, strcmp );
    for ( j=0; j<i; j++ )
        {
        xmstr = XmStringLtoRCreate( selList[j], XmSTRING_DEFAULT_CHARSET );
        XmListAddItem( list, xmstr, j+1 );
        XmStringFree( xmstr );
        }
    }

if ( global )
    destroy_global_varlist( varlist );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  lookup_global_varlist()
*
* MODULE FUNCTION:
*
*   This routine forms a list of the global variables for the current element.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

struct symbol_entry  *lookup_global_varlist()

{
    struct symbol_entry *temp, *head = NULL, *sym_entry;

    /*
     *  Initialize sym_entry to point to the root of the symbol table.
     */

    sym_entry = symbol_table;

    while ( sym_entry )
        {
        if ( sym_entry->se_type & VARIABLE )
            {

            /*
             *  only elements, intrinsic functions and variables are global; of
             *  these, only variables have the VARIABLE flag set. add this entry
             *  to list.
             */

            if ( temp = (struct symbol_entry *)malloc( sizeof( struct symbol_entry ) ) )
                {
                strcpy( temp->se_symbol, sym_entry->se_symbol);
                temp->se_type           =  sym_entry->se_type;
                temp->se_use_count      =  sym_entry->se_use_count;
                temp->se_num_dimensions = sym_entry->se_num_dimensions;
                temp->se_subs[0]        = sym_entry->se_subs[0];
                temp->se_subs[1]        = sym_entry->se_subs[1];
                temp->se_subs[2]        = sym_entry->se_subs[2];
                temp->se_subs[3]        = sym_entry->se_subs[3];

                temp->se_local_vars = NULL;
                }

            temp->se_next = head;
            head = temp;
            }
        sym_entry = sym_entry->se_next;
        }

    return( (struct symbol_entry *) head );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  valid_name()
*
* MODULE FUNCTION:
*
*   This routine determines if the name has only the proper type of characters
*   and number of brackets.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  valid_name( name, popup_type )

    char    name[];
    int     popup_type;

{
    int attribs,
        j,
        len,
        left_br_count = 0,
        rght_br_count = 0;

    /*
     *  If the variable is not a local "looking" variable but the popup thinks the
     *  variable was supposed to be a local, then error.
     */

    attribs = is_local( name );

    if ( attribs & LOCAL_VAR )
        if ( popup_type != LOCAL_VAR )
            return( ERR );
    else if ( attribs & WS_GLOBAL )
        if ( popup_type != WS_GLOBAL )
            return( ERR );
    else if ( attribs & WS_OBJECT )
        if ( popup_type != WS_OBJECT )
            return( ERR );
    else if ( attribs & GLOBAL_VAR )
        if ( popup_type != GLOBAL )
            return( ERR );

    /*
     *  If the variable name is empty, then error.
     */

    len = strlen( name );
    if (! len )
        return( ERR );

    /*
     *  Go through the variable name and look for an illegal character.  The first
     *  character should only be alphabetic, no underscores or numbers.
     */

    for ( j=0; j<len; j++ )
        {

        /*
         *  First character must be alphabetic.
         */

        if (! j )
            {
            if (! isalpha(name[j]) )
                return( ERR );
            }

        /*
         *  Remaining characters can be alpha or numeric or matrix identifier.
         */

        else
            if (! isalnum(name[j]) )
                if (! index("_[]",name[j]) )
                    return( ERR );
                else
                    /*
                     *  Record brackets.
                     */
                    {
                    if ( name[j] == '[' )
                        left_br_count++;
                    if ( name[j] == ']' )
                        rght_br_count++;
                    }
        }

    /*
     *  Do some checking on the brackets.
     */

    if ( rght_br_count != left_br_count )
        return( ERR );

    if ( (popup_type == WS_GLOBAL) || (popup_type == WS_OBJECT) )
        if ( rght_br_count > 1 )
            return( ERR );

    /*
     *  Variable must be okay, otherwise we wouldn't get this far.
     */

    return( OK );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  valid_num()
*
* MODULE FUNCTION:
*
*   This routine  determines if the number is well_formed.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  valid_num( text )

    char    *text;

{
    int j, dot = 0;

    /*
     *   a NULL string is not well-formed.
     */

    if ( !strlen(text) )
        return( 0 );

    for ( j= 0; j<strlen(text); j++ )
        {
        if ( text[j] == '.' )
            {

            /*
             *  >1 dot?
             */

            if ( dot )
                return( 0 );
            else dot = TRUE;
            }
        else if ( (text[j] == 'x') || (text[j] == 'X') )
            {

            /*
             *  hex number
             */

            if ( (!j) || (text[j-1] != '0') )
                return( 0 );
            }
        else if ( !isdigit(text[j]) )
            return( 0 );
        }
    return( 1 );
}
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    cbr_var_input.h
*
* FILE FUNCTION:
*
*   This file contains the global variables and function prototypes for cbr_var_input.c
*
*
* SPECIFICATION DOCUMENTS:
*
*   /home/project/3531/Docu/GCB.spec.doc
*
*
* FILE MODULES:
*
*   N/A
*
*****************************************<----->*****************************************/

/*
 *  cbr_var_input.c global variables.
 */

char    variable[MAX_TEXT];

int     rb_sym_type;

/*
 *  cbr_var_input.c function prototypes.
 */

void    just_mat_name();
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    colors.c
*
* FILE FUNCTION:
*
*   This file contains the routines which create and manipulate the display colors.
*
* SPECIFICATION DOCUMENTS:
*
*   /home/project/3531/Docu/GCB.spec.doc
*
*
* FILE MODULES:
*
*   build_color_popup()     - builds the popup to select palette item colors.
*   cbr_color_choice()      - handles button press events over a color choice.
*   cbr_color_done()        - applies the color choices to palette and work areas.
*   cbr_color_menu()        - pops up the color menu from the pulldown menu.
*   cbr_color_sym_choice()  - changes bitmap in color popup according to selection
*   get_colors()            - sets up the color map for gcb.
*
*****************************************<----->*****************************************/

#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/cursorfont.h>
#include <Xm/Xm.h>
#include <Xm/PushB.h>
#include <Xm/RowColumn.h>

#include "gcb.h"
#include "cbr.h"
#include "widgets.h"
#include "menu.h"
#include "pixmaps.h"
#include "constants.h"
#include "images.h"

extern char     *palette_items[];

/*
 *  header string for colors popup.
 */

char color_string[] = "Select the item whose colors you wish to change. Then select the
background or foreground.  Next, select the color.  \nTo change the color of the palette
 item and corresponding symbols, click Apply. To restore the original colors, \nclick Re
set.";

static char     *cnames[] = {
        "white",
        "gray",
        "khaki",
        "gold",
        "yellow",
        "yellow green",
        "green",
        "aquamarine",
        "sky blue",
        "dark slate blue",
        "brown",
        "black"
        };

/*
 *  vars for storing current color choices
 */

static unsigned long    btn_fore, btn_back;
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  build_color_popup()
*
* MODULE FUNCTION:
*
*   This routine builds the popup from which the user selects palette item colors.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  build_color_popup( parent )

    Widget      parent;

{
    int             n, i;
    Arg             args[10];
    XImage          *image;
    Pixmap          pixmap;
    Widget          button;

    dlg_color = cr_popup( NULLS, parent, "Select Colors" );
    FormW = cr_form( NULLS, dlg_color, NULL, NULL );
    set_attribs( FORM, FormW, 775, 400, XmRESIZE_NONE );

    lbl_color_header = cr_label( NULLS, FormW, color_string, 0, 3, 20, 5, 95 );

    /*
     *  header is left-aligned.
     */

    XtSetArg( args[0], XmNalignment, XmALIGNMENT_BEGINNING );
    XtSetValues( lbl_color_header, args, 1 );

    /*
     *  create rowcol to hold palette name items
     */

    n = 0;
    XtSetArg( args[n], XmNpacking, XmPACK_COLUMN ); n++;
    XtSetArg( args[n], XmNnumColumns, 1 ); n++;
    XtSetArg( args[n], XmNentryAlignment, XmALIGNMENT_CENTER ); n++;
    XtSetArg( args[n], XmNorientation, XmHORIZONTAL ); n++;
    rc_color_iname = XmCreateRowColumn( FormW, "rc_color_iname", args, n );
    XtManageChild( rc_color_iname );
    set_position( rc_color_iname, 25, IGNORE, 5, IGNORE );

    for ( i = 0; i < 10; i++ )
        button = cr_command( NULLS, rc_color_iname, palette_items[i],
            cbr_color_sym_choice, i );
    button = cr_command( NULLS, rc_color_iname, "All",
            cbr_color_sym_choice, 11 );

    image = (XImage *)CreateDefaultImage( box_bits, 16, 16 );
    XmInstallImage( image, "box" );

    /*
     *  create rowcol to hold colors
     */

    n = 0;
    XtSetArg( args[n], XmNpacking, XmPACK_COLUMN ); n++;
    XtSetArg( args[n], XmNnumColumns, 1 ); n++;
    XtSetArg( args[n], XmNentryAlignment, XmALIGNMENT_CENTER ); n++;
    XtSetArg( args[n], XmNorientation, XmHORIZONTAL ); n++;
    rc_color = XmCreateRowColumn( FormW, "rc_color", args, n );
    XtManageChild( rc_color );
    set_position( rc_color, 60, IGNORE, 5, IGNORE );

    /*
     *  find pixmap for each button, set callback function, and
     *  create button.
     */

    for ( i=0; i < MAX_COLORS-1; i++ )
        {
        pixmap = XmGetPixmap ( XtScreen(rc_color), "box", colors[i], 0 );

        /*
         *  create push button
         */

        n = 0;
        XtSetArg( args[n], XmNlabelType, XmPIXMAP ); n++;
        XtSetArg( args[n], XmNlabelPixmap, pixmap ); n++;
        XtSetArg( args[n], XmNwidth, 16 ); n++;
        XtSetArg( args[n], XmNheight, 16 ); n++;
        button = XmCreatePushButton( rc_color, NULLS, args, n );
        XtManageChild( button );

        XtAddCallback( button, XmNactivateCallback, cbr_color_choice, i );
        }

    /*
     *  create rowcol to hold palette items
     */

    n = 0;
    XtSetArg( args[n], XmNpacking, XmPACK_COLUMN ); n++;
    XtSetArg( args[n], XmNnumColumns, 1 ); n++;
    XtSetArg( args[n], XmNentryAlignment, XmALIGNMENT_CENTER ); n++;
    XtSetArg( args[n], XmNorientation, XmHORIZONTAL ); n++;
    rc_color_template = XmCreateRowColumn( FormW, "rc_color_template", args, n );
    XtManageChild( rc_color_template );
    set_position( rc_color_template, 45, IGNORE, 75, IGNORE );

    /*
     *  find pixmap for begin button, create template button.
     */

    pixmap = XmGetPixmap ( XtScreen(rc_color), "BEGIN",
        colors[get_my_foreground(BEGIN)], colors[get_my_background(BEGIN)] );
    n = 0;
    XtSetArg( args[n], XmNlabelType, XmPIXMAP ); n++;
    XtSetArg( args[n], XmNlabelPixmap, pixmap ); n++;
    XtSetArg( args[n], XmNwidth, 64 ); n++;
    XtSetArg( args[n], XmNheight, 64 ); n++;
    XtSetArg( args[n], XmNuserData, 0 ); n++;
    btn_color_template = XmCreatePushButton( rc_color_template, NULLS, args, n );
    XtManageChild( btn_color_template );
```

```
    rb_color = cr_radio_box( NULLS, FormW, XmVERTICAL );

    cr_separator( NULLS, FormW, 82, IGNORE, 1, 99 );

    CancelW         = cr_command( NULLS, FormW, "Close", cbr_color_done, 3 );
    DoneW           = cr_command( NULLS, FormW, "Apply", cbr_color_done, 1 );
    ResetW          = cr_command( NULLS, FormW, "Reset", cbr_color_done, 2 );
    HelpW           = cr_command( NULLS, FormW, "Help",  cbr_help,      SET_COLORS );

    set_position( rb_color,         40, IGNORE,  5, IGNORE );
    set_position( CancelW,          90, IGNORE,  5, IGNORE );
    set_position( DoneW,            90, IGNORE, 30, IGNORE );
    set_position( ResetW,           90, IGNORE, 58, IGNORE );
    set_position( HelpW,            90, IGNORE, 83, IGNORE );

    tgl_color_fore = cr_toggle( NULLS, rb_color, "Foreground", NULL, 0, 0 );
    tgl_color_back = cr_toggle( NULLS, rb_color, "Background", NULL, 0, 0 );

    btn_fore = get_my_foreground( BEGIN );
    btn_back = get_my_background( BEGIN );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_color_choice()
*
* MODULE FUNCTION:
*
*   This routine handles events when the user pushes a mouse button over a
*   color choice.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc cbr_color_choice ( w, color, call_data )

    Widget  w;
    int     color;
    caddr_t call_data;

{
    Pixmap      pixmap;
    Arg         args[1];
    int         sym;

    /*
     *  retrieve symbol # whose colors we are changing
     */

    XtSetArg( args[0], XmNuserData, &sym );
    XtGetValues( btn_color_template,args, 1 );

    /*
     *  if user selected all, set template to BEGIN.
     */

    if ( sym > (NUM_PALETTE-1) )
        sym = 0;

    if ( XmToggleButtonGetState(tgl_color_fore) )

        /*
         *  foreground tgl is set
         */

        {
        pixmap = XmGetPixmap ( XtScreen(rc_color), palette_items[sym],
            colors[color], colors[btn_back] );

        /*
         *  Remember most recent foreground color choice.
         */

        btn_fore = color;

        }
    else

        /*
         *  background tgl is set
```

```
         */

        {
        pixmap = XmGetPixmap ( XtScreen(rc_color), palette_items[sym],
            colors[btn_fore], colors[color] );

        /*
         *  Remember most recent foreground color choice.
         */

        btn_back = color;
        }

    XtSetArg( args[0], XmNlabelPixmap, pixmap );
    XtSetValues( btn_color_template, args, 1 );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_color_menu()
*
* MODULE FUNCTION:
*
*   This routine pops up the color menu from the pulldown menu.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc cbr_color_menu ( w, client_data, call_data )

    Widget  w;
    caddr_t client_data;
    caddr_t call_data;

{
    if ( (Color) && (Mode == EditSymbol) )
        {
           arm_tgl( tgl_color_back );
        disarm_tgl( tgl_color_fore );
        XtManageChild( dlg_color );
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_color_sym_choice()
*
* MODULE FUNCTION:
*
*   This routine changes the bitmap in the color popup according to the selected
*   palette item.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc cbr_color_sym_choice ( w, client_data, call_data )

    Widget  w;
    caddr_t client_data;
    caddr_t call_data;

{
    Pixmap      pixmap;
    Arg         args[2];
    int         sym = (int)client_data;

    if ( sym > (NUM_PALETTE-1) )

        /*
         *  if 'all' is selected, the representative pixmap is BEGIN
         */

        sym = 0;

    btn_fore = get_my_foreground( sym );
    btn_back = get_my_background( sym );

    /*
     *  retrieve palette item pixmap from Motif cache.
     */

    pixmap = XmGetPixmap ( XtScreen(rc_color), palette_items[sym],
        colors[btn_fore], colors[btn_back] );
    XtSetArg( args[0], XmNlabelPixmap, pixmap );
    XtSetArg( args[1], XmNuserData, (int)client_data );

    /*
     *  set template's user data to the symbol number selected.
     */

    XtSetValues( btn_color_template, args, 2 );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_color_done()
*
* MODULE FUNCTION:
*
*   This routine  applies the color choices to the palette and work areas.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc cbr_color_done ( w, client_data, call_data )

    Widget      w;
    int         client_data;
    caddr_t     call_data;

{
    int         i, sym;
    Arg         args[1];
    Pixmap      pixmap;

    if ( client_data == 3 )

        /*
         *  cancel.
         */

        XtUnmanageChild( dlg_color );

    else if ( client_data == 2 )

        /*
         *  reset.
         */

        {
        XtSetArg( args[0], XmNuserData, &sym );
        XtGetValues( btn_color_template, args, 1 );

        cbr_color_sym_choice( w, sym, call_data );
        }

    else
        {
        XtSetArg( args[0], XmNuserData, &sym );
        XtGetValues( btn_color_template,args, 1 );

        if ( sym < NUM_PALETTE )
            {

            set_my_foreground( sym, btn_fore );
            set_my_background( sym, btn_back );

            /*
             *  change palette item's colors to reflect btn_color_template
             */
```

```
        pixmap = XmGetPixmap ( XtScreen(rc_color), palette_items[sym],
            colors[btn_fore], colors[btn_back] );
        XtSetArg( args[0], XmNlabelPixmap, pixmap );
        XtSetValues( Palette[sym], args, 1 );
        redraw_sym_type( sym );
        }
    else

        /*
         *  set fore and backs of all palette items
         */

        for ( i=0; i < NUM_PALETTE; i++ )
            {
            set_my_foreground( i, btn_fore );
            set_my_background( i, btn_back );

            /*
             *  change palette item's colors to reflect btn_color_template
             */

            pixmap = XmGetPixmap ( XtScreen(rc_color), palette_items[i],
                colors[get_my_foreground(i)], colors[get_my_background(i)] );
            XtSetArg( args[0], XmNlabelPixmap, pixmap );
            XtSetValues( Palette[i], args, 1 );
            redraw_sym_type( i );
            }
    }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  get_colors()
*
* MODULE FUNCTION:
*
*   This routine sets up the color map for gcb.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  get_colors()

{
    Colormap    cmap;
    XColor      exact_def;
    int         i;

    /*
     *  get X's default color map
     */

    cmap = DefaultColormap( display, DefaultScreen(display) );
    if (Color)
        {

        /*
         *  allocate the colors in the cnames array.
         */

        for ( i=0; i<MAX_COLORS-1; i++ )
            {
            if ( !XParseColor(display, cmap, cnames[i], &exact_def) )
                {
                elog( 1,"getcolors: color name %s not in db", cnames[i] );
                exit( 0 );
                }
            if ( !XAllocColor(display, cmap, &exact_def) )
                {
                elog( 1,"getcolors: all color cells allocated" );
                exit( 0 );
                }
            colors[i] = exact_def.pixel;
            }

        /*
         *  allocate red for audit
         */

        if ( !XParseColor(display, cmap, "red", &exact_def) )
            {
            elog( 1,"getcolors: color name %s not in db", "red" );
            exit( 0 );
            }
        if (! XAllocColor(display, cmap, &exact_def) )
            {
            elog(1,"getcolors: all color cells allocated");
            exit( 0 );
```

```
            }
        colors[MAX_COLORS-1] = exact_def.pixel;
        Red = MAX_COLORS-1;
        }
    else
        {

        /*
         *  mono
         */

        colors[0] = WhitePixel( display, DefaultScreen(display) );
        colors[1] = BlackPixel( display, DefaultScreen(display) );
        }

    /*
     *  color for inactive expression creation buttons
     */
    insensitive_color = colors[1];
}
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    comp_file.c
*
*
* FILE FUNCTION:
*
*   This file contains the routines which process the Comp file information.
*   This file contains the routines which create the maintain Comp file
*   popups and also contains the routines which read and write Comp disk
*   files.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /home/project/3531/Docu/GCB.spec.doc
*
*
* FILE MODULES:
*
*   build_cre_comp_popup()   - build the Create Comp popup
*   build_purpose_popup()    - build the Comp/Element popup for Purpose editing
*   build_sel_comp_popup()   - build the Select Comp popup
*   cbr_comp_selected()      - process the Selection list during Select Comp
*   cbr_cre_comp()           - display the Create Comp popup
*   cbr_new_comp()           - process the Create Comp popup
*   cbr_sel_comp()           - process the CANCEL button during Select Comp
*   del_element_from_comp()  - remove Element reference from Comp file
*   init_comp_vars()         - initialize Comp level global variables
*   load_comp_list()         - load Selection list widget with Comp dir names
*   read_comp_file()         - read Comp file information
*   read_comp_list()         - read Element file names from Comp file
*   update_comp_file()       - update and write a Comp file
*   valid_comp_dir_name()    - make sure filename is a Comp directory name
*   valid_comp_name()        - make sure filename is a Comp file
*   write_comp_file()        - write out Comp file to disk
*
*****************************************<----->*****************************************/

#include <stdio.h>
#include <sys/file.h>
#include <sys/types.h>
#include <dirent.h>

#include <X11/Intrinsic.h>
#include <Xm/Xm.h>

#include "gcb.h"
#include "cbr.h"
#include "widgets.h"
#include "comp_file.h"
#include "element_file.h"
#include "constants.h"
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  build_cre_comp_popup()
*
* MODULE FUNCTION:
*
*   This routine builds the popup which allows the user to create a Comp
*   file.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

Widget  build_cre_comp_popup( parent )

    Widget  parent;

{
    Arg     args[1];

    dlg_cre_comp = cr_popup( NULLS, parent, "Create Comp" );

    FormW = cr_form( NULLS, dlg_cre_comp, NULL, NULL );
    set_attribs( FORM, FormW, 475, 400, XmRESIZE_NONE );

    cr_label( NULLS, FormW, "Comp Name:", 0, 7, 12, 9, IGNORE );
    txt_cre_comp_name = cr_text (NULLS, FormW, NULL, NULL, NULLS, FALSE, 1, 37);

    XtAddCallback( txt_cre_comp_name, XmNactivateCallback,     cbr_text_proc, 2);
    XtAddCallback( txt_cre_comp_name, XmNmodifyVerifyCallback,cbr_text_proc, 2);
    XtSetArg( args[0], XmNtraversalOn, True );
    XtSetValues( txt_cre_comp_name, args, 1 );

    cr_label(NULLS,FormW,"Root Element Name:",0,20,25,9,IGNORE);

    txt_cre_comp_re_name = cr_text(NULLS, FormW, NULL, NULL, NULLS, FALSE, 1, 37);
    XtAddCallback( txt_cre_comp_re_name, XmNactivateCallback, cbr_text_proc, 3);
    XtSetArg( args[0], XmNtraversalOn, True );
    XtSetValues( txt_cre_comp_re_name, args, 1 );

    cr_label(NULLS, FormW, "Purpose:", 0, 35, 40, 9, IGNORE);
    scr_cre_comp     = cr_scr_text(NULLS, FormW, 10, True, 275, NULL, NULL);

    cr_separator( NULLS, FormW, 87, IGNORE, 1, 99 );

    DoneW   = cr_command( NULLS,FormW, "Create", cbr_new_comp, DONE );
    CancelW = cr_command( NULLS,FormW, "Cancel", cbr_new_comp, CANCEL );
    HelpW   = cr_command( NULLS,FormW, "Help",   cbr_help,     CREATE_COMP );

    set_position( XtParent(scr_cre_comp),   35, IGNORE, 35, IGNORE );
    set_position( txt_cre_comp_name,         7, IGNORE, 35, IGNORE );
    set_position( txt_cre_comp_re_name,     20, IGNORE, 35, IGNORE );
    set_position( CancelW,                  92, IGNORE,  3, IGNORE );
    set_position( DoneW,                    92, IGNORE, 41, IGNORE );
    set_position( HelpW,                    92, IGNORE, 79, IGNORE );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  build_purpose_popup()
*
* MODULE FUNCTION:
*
*   This routine builds the popup which allows the user to edit a Comp or
*   Element file purpose text string.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

Widget  build_purpose_popup( parent )

    Widget parent;

{
    int     n;


    dlg_purpose = cr_popup( NULLS, parent, "Input Purpose" );

    FormW = cr_form( NULLS, dlg_purpose, NULL, NULL );
    set_attribs( FORM, FormW, 475, 300, XmRESIZE_NONE );

    cr_label( NULLS, FormW, "Purpose:  ", 0, 10,IGNORE,9,IGNORE);
    scr_purpose = cr_scr_text( NULLS, FormW, 10, True, 265, NULL, NULL);

    CancelW = cr_command( NULLS, FormW, "Cancel", cbr_pos_done, CANCEL );
    DoneW   = cr_command( NULLS, FormW, "Done",   cbr_pos_done, DONE );
    HelpW   = cr_command( NULLS, FormW, "Help",   cbr_help,     COMP_PURP_HELP);

    cr_separator( NULLS, FormW, 80, IGNORE, 1, 99 );

    set_position( XtParent(scr_purpose),10, IGNORE, 25, IGNORE );
    set_position( CancelW,              88, IGNORE,  5, IGNORE );
    set_position( DoneW,                88, IGNORE, 40, IGNORE );
    set_position( HelpW,                88, IGNORE, 75, IGNORE );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  build_sel_comp_popup()
*
* MODULE FUNCTION:
*
*   This routine builds the popup which allows the user to select a Comp
*   from a selection list for the current Position.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

Widget  build_sel_comp_popup( parent )

    Widget parent;

{
    int     n;


    dlg_sel_comp = cr_popup( NULLS, parent, "Select Comp" );

    FormW = cr_form( NULLS, dlg_sel_comp, NULL, NULL );
    set_attribs( FORM, FormW, 450, 400, XmRESIZE_NONE );

    cr_label( NULLS, FormW, "Comp List:", 0,10,15,9, IGNORE);
    list_sel_comp     = cr_list ( NULLS, FormW, 14, 215 );
    XtAddCallback( list_sel_comp, XmNbrowseSelectionCallback, cbr_comp_selected, NULL );

    cr_separator( NULLS, FormW, 85, IGNORE, 1, 99 );

    CancelW = cr_command( NULLS,FormW, "Cancel",cbr_sel_comp, CANCEL );
    HelpW   = cr_command( NULLS,FormW, "Help",  cbr_help, SELECT_COMP);

    set_position( XtParent(list_sel_comp),  10, IGNORE, 35, IGNORE );
    set_position( CancelW,                  91, IGNORE,  5, IGNORE );
    set_position( HelpW,                    91, IGNORE, 75, IGNORE );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_comp_selected()
*
*
* MODULE FUNCTION:
*
*   This routine processes the user's button presses during the selection of
*   a Comp.  This routine is called when the user selects a Comp from the
*   Selection list.  This routine only processes the Selection list callbacks.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  cbr_comp_selected( w, client_data, call_data )

    Widget                  w;
    caddr_t                 *client_data;
    XmListCallbackStruct    *call_data;

{
    char *string;

    /*
     *  Get the string the user picked in the selection box.
     */

    XmStringGetLtoR( call_data->item, XmSTRING_DEFAULT_CHARSET, &string );

    busy( dlg_sel_comp, TRUE );

    /*
     *  See if the current file has been updated and should be saved.
     */

    if ( SaveNeeded )
        save_curr_element();

    reinit_element_vars();
    init_comp_vars();

    /*
     *  Move down into the comp directory.
     */

    chdir ( PositionPath );

    strcpy( CompDir,   string );
    strcat( CompDir,   ".DIR" );
    chdir ( CompDir );

    strcpy( CompFile,  string );
    strcpy( GCompFile, string );
    strcat( GCompFile, CMP_EXT);
    read_comp_file();

    /*
     *  Update the onscreen status indicators and then take down the
     *  popup.
     */

    upd_pos_panel( COMP_PURPOSE );
    upd_mode_panel();

    XtUnmanageChild(dlg_sel_comp);
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_cre_comp()
*
* MODULE FUNCTION:
*
*   This routine pops up the create Comp popup.  This routine is a callback
*   which is used by the menu system to display the create Comp popup to the
*   user.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_cre_comp( w, client_data, call_data )

    Widget  w;
    caddr_t client_data,
            call_data;

{
    if ( Mode == EditSymbol )
        {
        XtManageChild( dlg_cre_comp );
        busy( dlg_cre_comp, FALSE );
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_new_comp()
*
* MODULE FUNCTION:
*
*   This routine responds to the user's button presses in the Create Comp
*   popup.  This routine processes the DONE and CANCEL buttons.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_new_comp( w, client_data, call_data )

    Widget  w;
    int     client_data;
    caddr_t call_data;

{
    FILE    *fp;
    int     key,
            pos,
            sym_size = 0;
    char    tempPath[MAX_PATH];


    if ( client_data == DONE )
        {

        /*
         *  Make sure the user entered a Comp name and Root Element name.
         */

        if (! strlen(XmTextGetString(txt_cre_comp_name)) )
            {
            user_ack("Please enter a new Comp name");
            return;
            }

        if (! strlen(XmTextGetString(txt_cre_comp_re_name)) )
            {
            user_ack("Please enter a Root Element name");
            return;
            }

        /*
         *  See if the current file has been updated and should be saved.
         */

        busy( dlg_cre_comp, TRUE );

        if ( SaveNeeded )
            save_curr_element();

        /*
         *  Clear work area if needed.
         */
```

```
reinit_element_vars();
  init_comp_vars();

strcpy( CompFile,     (char *) XmTextGetString(txt_cre_comp_name) );
strcpy( RootElement,  (char *) XmTextGetString(txt_cre_comp_re_name) );

/*
 *  Create the Comp directory string and then create the new comp
 *  subdirectory.
 */

strcpy( CompDir,  CompFile );
strcat( CompDir,  ".DIR"   );
strcpy( tempPath, PositionPath );
strcat( tempPath, "/"      );
strcat( tempPath, CompDir  );

if ( mkdir(tempPath,0777) )
    {
    ValidComp = FALSE;
    user_ack("Couldn't create the Comp directory, Comp not created.");
    elog(1,"cbr_new_comp: Couldn't create the Comp directory: %s",CompDir);
    return;
    }

/*
 *  Move into the new Comp directory and create the Comp file.
 */

chdir( tempPath );

strcpy( GCompFile, CompFile );
strcat( GCompFile, CMP_EXT  );

if (! access(GCompFile,R_OK) )
    {
    user_ack("Couldn't open the new comp file, file already exists");
    return;
    }

strcpy( CompPurpose, (char *)XmTextGetString(scr_cre_comp) );

/*
 *  Create the new comp file.
 */

if (! (fp = fopen(GCompFile,"w")) )
    {
    user_ack("Couldn't open the Comp file for writing, Comp not created");
    elog(1,"Couldn't open the Comp file, Comp %s not created",GCompFile);
    return;
    }

/*
 *  Save the purpose string and the root element name.  Save the end-
 *  of-elements indicator.
 */

save_element_str( fp, CompPurpose );

fprintf( fp, "\n%c %s\n", 'E', RootElement );
fprintf( fp, "2 22\n" );

/*
 *  Save the symbol table size and then close the file.
 */

fwrite( &sym_size, sizeof(int), 1, fp );
fclose( fp );

/*
 *  Mark the comp as valid and update the status indicators, then
 *  take down the popup.
 */

ValidComp = TRUE;
upd_pos_panel( COMP_PURPOSE );
upd_mode_panel();

XtUnmanageChild( dlg_cre_comp );
}

/*
 *  User doesn't want to create a new comp, take down the popup.
 */

else if ( client_data == CANCEL )
    {
    if (! ValidComp )
        {
        CompFile[0]  = NULL;
        GCompFile[0] = NULL;
        }
    XtUnmanageChild( dlg_cre_comp );
    }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_sel_comp()
*
* MODULE FUNCTION:
*
*   This routine processes the user's button presses in the Select Comp
*   popup.  This routine processes the CANCEL button only.  This routine is
*   also called by the menu system to display the Select Comp popup.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_sel_comp( w, client_data, call_data )

    Widget  w;
    int     client_data;
    caddr_t call_data;

{
    if (Mode != EditSymbol)
        return;

    /*
     *  User wants to abort from selecting a comp, take down the popup.
     */

    if ( client_data == CANCEL )
        {
        XtUnmanageChild( dlg_sel_comp );
        return;
        }

    /*
     *  User has selected Select Comp from the menu system.  Show the Select
     *  Comp popup.
     */

    if ( client_data == MANAGE )
        {

        /*
         *  Load the selection list with the comp names.
         */

        if (! load_comp_list() )
            {
            XtManageChild( dlg_sel_comp );
            busy( dlg_sel_comp, FALSE );
            }
        return;
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  del_element_from_comp()
*
* MODULE FUNCTION:
*
*   This routine is used to remove a reference to an Element file when the user
*   chooses to delete an Element file.  The Element file name is removed from
*   the Comp file.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  del_element_from_comp( element, type )
{
    FILE *cfp;
    char elementList[MAX_FILES][MAX_NAME],
           typeList[MAX_FILES],
         purposeStr[MAX_PURPOSE];
    int  i,
         lsize = 0;


    /*
     *  Open the Comp file.  Read the comp purpose string.
     */

    if (! (cfp = fopen(GCompFile,"r")) )
        {
        user_ack("Couldn't open the Comp file to remove Element name");
        elog(1,"Couldn't open Comp file: %s for updating to remove element", GCompFile);
        return;
        }

    read_element_str( cfp, purposeStr );
    fscanf( cfp, " \n" );

    /*
     *  While not end of element list, read the Element names and insert them into
     *  the list.  Don't insert the name of the element we want to delete from the list.
     */

    while ( fscanf(cfp,"%c%s%*c",&typeList[lsize],elementList[lsize]) != EOF )
        {
        if ( typeList[lsize] == 'Z' )
            break;
        if ( strcmp(elementList[lsize],element) )
            lsize++;
        else
            {
            if ( type == LIB )
                if ( typeList[lsize] != 'L' )
                    lsize++;
            if ( type == ELEMENT )
                if ( typeList[lsize] != 'E' )
                    lsize++;
            }
        }
```

```
    fclose( cfp );

    /*
     *  The comp file has been read and the element name to delete has
     *  been removed from the list, now write the comp file back to disk.
     */

    write_comp_file( GCompFile, lsize, purposeStr, typeList, elementList, NULL );
}
```

```
/*****************************************<----->*************************************
*
* MODULE NAME:  init_comp_vars()
*
*
* MODULE FUNCTION:
*
*   This routine initializes the various Comp related variables.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                                  Release 1.02 - 08/28/91
*
*****************************************<----->*************************************/

void  init_comp_vars()
{
    int i;


    ValidComp    = FALSE;
    CompFile[0]  = NULL;
    GCompFile[0] = NULL;

    for ( i=0; i<MAX_PURPOSE; i++ )
        CompPurpose[i] = ' ';
    CompPurpose[0] = NULL;

    symbol_table_init();
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  load_comp_list()
*
* MODULE FUNCTION:
*
*   This routine loads the selection list widget of the Select Comp popup with
*   the Comp directory names from the current Position file directory.  Comp
*   directories are identified by their .DIR extension.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  load_comp_list()
{
    DIR             *dir;
    char            *cptr,
                    *nptr;
    struct dirent   *dp;
    int             cnt,
                    i;
    XmString        xmstr;


    XmListDeleteAllItems( list_sel_comp );

    /*
     *  Read the Position directory.
     */

    if ((dir = opendir(PositionPath)) == NULL)
        {
        user_ack("Error: couldn't open Position directory");
        elog(1,"load_comp_list() - couldn't read directory");
        return( ERR );
        }

    /*
     *  Insert only Comp directory names into the list.
     */

    for ( cnt=0, dp=readdir(dir); dp!=NULL; dp=readdir(dir) )
        {
        strcpy( selList[cnt], dp->d_name );
        if (! valid_comp_dir_name(selList[cnt]) )
            {
            nptr =  selList[cnt];
            cptr = &selList[cnt][strlen(nptr)-4];
            *cptr = NULL;
            cnt++;
            }
        if ( cnt == MAX_FILES )
            {
            user_ack("Error: exhausted file list - notify developer");
            elog(1,"load_comp_list: exhausted file list");
            return( ERR );
            }
        }

    /*
     *  Sort the list and install the list into the selection widget.
     */

    if ( cnt > 0 )
        {
        qsort( selList, cnt, MAX_NAME, strcmp );
        for (i=0; i<cnt; i++)
            {
            xmstr = XmStringCreate( selList[i], XmSTRING_DEFAULT_CHARSET );
            XmListAddItem( list_sel_comp, xmstr, i+1 );
            XmStringFree( xmstr );
            }
        }

    return( OK );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  read_comp_file()
*
* MODULE FUNCTION:
*
*   This routine reads the comp file.  The list of Comp elements is NOT read in.
*   The list of element names contained in this comp is read in when the selection
*   list is shown to the user.  Only the Root Element name is read by this routine.
*   The symbol table is also read.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  read_comp_file()
{
    FILE    *fp;
    char    type_char;
    int     rc;


    if (! (fp = fopen(GCompFile,"r"))  )
        {
        user_ack("Error: could not open comp file");
        return( ERR );
        }

    read_element_str( fp, CompPurpose );

    /*
     *  Read the first element which is the root element.
     */

    rc = fscanf( fp," \n%c%s%*c", &type_char, RootElement );

    /*
     *  Read the rest of the element file names, but throw them away.
     */

    while ( fscanf(fp,"%c%*s%*c",&type_char) != EOF )
        if ( type_char == 'Z' )
            break;

    /*
     *  Read the symbol table.
     */

    if ( symbol_table_restore(fp) )
        {
        user_ack("Couldn't restore the symbol table from the Comp File");
        elog(1,"Couldn't restore the symbol table from: %s", GCompFile);
        user_ack("Unrecoverable error - GCB exiting");
        exit( ERR );
        }

/*DEBUG*/
print_symbol_table();

    /*
     *  Close the Comp file.
     */

    fclose( fp );
    ValidComp = TRUE;
    return( OK );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  read_comp_list()
*
* MODULE FUNCTION:
*
*   This routine reads in a list of element names of one of two types, either
*   comp elements or library elements.  The element names are read from a
*   previously opened Comp file.  The number of element names which are inserted
*   into the list is returned.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  read_comp_list( fp, type, list )

    FILE *fp;
    char list[][MAX_NAME];

{
    int     cnt = 0;
    char    type_char;

    /*
     *  Read the purpose string and the element names from the Comp file.
     */

    read_element_str( fp, Purpose );
    fscanf( fp, " \n" );

    while ( fscanf(fp,"%c%s%*c",&type_char,list[cnt]) != EOF )
        {
        if ( type_char == 'Z' )
            break;
        if ((type_char == 'L') && (type == LIB))
            cnt++;
        else if ((type_char == 'E') && (type == ELEMENT))
            cnt++;
        }

    return( cnt );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  update_comp_file()
*
* MODULE FUNCTION:
*
*   This routine updates the comp file.  It reads the element names from the
*   comp file, traverses from the root element to determine if there are any
*   comps which are referenced but not included, and then makes a call to
*   create a shiny new comp file.  If the root_name or purpose parameters are
*   NULL pointers, then the value in the existing comp file is used, if these
*   parameters actually point to data, then the value of the parameters is used.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  update_comp_file( comp_file, purpose, rootElem, newElem, newType  )

    char    *comp_file,
            *newElem,
            *newType,
            *purpose,
            *rootElem;

{
    FILE *cfp,
         *efp;

    char epath[MAX_PATH],
         elementName[MAX_NAME],
         elementList[MAX_FILES][MAX_NAME],
            typeList[MAX_FILES],
          purposeStr[MAX_PURPOSE];

    int  elementType,
         i,
         lsize = 0,
         ptr,
         rc,
         symbol_type;

    /*
     *  Open the Comp file.
     */

    if (! (cfp = fopen(comp_file,"r")) )
        {
        user_ack("Couldn't open the Comp file for reading, Comp file not updated");
        elog(1,"Couldn't open the Comp file for reading: %s", comp_file);
        return;
        }

    /*
     *  Read the purpose string from the Comp file.  If we need to update the
     *  purpose string, throw away the one in the Comp file.
     */

    if ( purpose )
```

```
        read_element_str( cfp, NULL );
    else
        read_element_str( cfp, purposeStr );

    fscanf( cfp, " \n" );

    /*
     *  While not EOF, read the Element names and insert them into
     *  the list.  Ignore any duplicates.
     */

    while ( fscanf(cfp,"%c%s%*c",&typeList[lsize],elementList[lsize]) != EOF )
        {
        /*
         *  Check for the end of the element list.
         */

        if ( typeList[lsize] == '2' )
            break;

        /*
         *  Check for the comp file entry to be the same as the specified root name.
         */

        if ( rootElem && typeList[lsize]=='E' && !strcmp(elementList[lsize],rootElem) )
            continue;

        /*
         *  Check for the comp file entry to be the same as the specified new
         *  element name.
         */

        if ( newElem && typeList[lsize]== *newType &&
            (!strcmp(elementList[lsize],newElem)) )
            continue;

        /*
         *  Check the entry read from the comp file to make sure it is not
         *  already in the list.
         */

        i = 0;
        while ( i < lsize )
            {
            if ( typeList[lsize] == typeList[i] )
                if (! strcmp(elementList[lsize],elementList[i]) )
                    break;
            i++;
            }
        if ( i == lsize )
            lsize++;
        }

    fclose( cfp );

    /*
     *  Add the new element to the list.
     */

    if ( newElem )
        {
        strcpy( elementList[lsize], newElem );
        typeList[lsize] = *newType;
        lsize++;
        }

    /*
     *  Open each of the element files in the list, locate any calls to other
     *  element files.
     */

    for ( ptr=0; ptr<lsize; ptr++ )
        {
        if ( typeList[ptr] == 'L' )
            {
            strcpy( epath, LibPath );
            strcat( epath, "/" );
            strcat( epath, elementList[ptr] );
            strcat( epath, EL_EXT );
            }
        else
            {
            strcpy( epath, elementList[ptr] );
            strcat( epath, EL_EXT );
            }

        /*
         *  Open the element file.
         */

        if (! (efp = fopen(epath,"r")) )
            {
            elog(3,"Couldn't open an Element file during Comp File update: %s", epath);
            continue;
            }

        /*
         *  Locate a symbol which will contain an element file name.
         */

        while ((rc = fscanf(efp,"%d",&symbol_type)) != EOF )
            {

            /*
             *  If we don't get an integer, dump the rest of the line.  If we
             *  get a LINE SEGMENT indicator, we are finished with the current
             *  file because the line info follows the symbol info.
             */

            if ( rc == 0 )
                {
                fscanf( efp, "%*[^\n]" );
                continue;
                }

            if ( symbol_type == SEGMENT_KEY )
                break;

            /*
             *  See if we have a symbol which may contain a reference to an
             *  element file.
             */

            if ( symbol_type == GOTO )
                {
                fscanf( efp,"%*u %*d %*d %*d %*d %*d %*d %*d %*d %*d" );
                read_element_str( efp, elementName );
                fscanf( efp, "%*u %*u %d", &elementType );
```

```
            fscanf( efp, "%*[^\n]" );
            }
        else
            {
            fscanf( efp, "%*[^\n]" );
            continue;
            }

        /*
         *  Loop through the list, if the new element name is already in
         *  the list, ignore it, otherwise add the new element name and
         *  update the list size.
         */

        i = 0;
        while ( i<lsize )
            if (! strcmp(elementList[i],elementName) )
                {
                if ((typeList[i] == 'L') && (elementType == LIB))
                    break;
                else if ((typeList[i] == 'E') && (elementType == ELEMENT))
                    break;
                else
                    i++;
                }
            else
                i++;

        /*
         *  If we are at the end of the list, this name must be a new
         *  one so add it to the list.
         */

        if ( i == lsize )
            {
            strcpy( elementList[lsize], elementName );
            if ( elementType == LIB )
                typeList[lsize] = 'L';
            else
                typeList[lsize] = 'E';
            lsize++;
            }
        }

    /*
     *  Close the current file, go get another.
     */

    fclose( efp );
    }

/*
 *  Write out our new, updated Comp file.
 */

if ( purpose )
    write_comp_file( comp_file, lsize, purpose, typeList, elementList, rootElem );
else
    write_comp_file( comp_file, lsize, purposeStr, typeList, elementList, rootElem )
;
return;
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  valid_comp_dir_name()
*
* MODULE FUNCTION:
*
*   This routine validates the Comp directory name.  Valid Comp directories can
*   be identified by a ".DIR" extension.  This routine was designed to be fast
*   and robust.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  valid_comp_dir_name( name )

    char *name;

{
    char *ptr;


    if ( strlen(name) < 5 )
        return( ERR );

    ptr = &name[strlen(name)-1];

    if ( *ptr != 'R' )
        return( ERR );
    ptr--;

    if ( *ptr != 'I' )
        return( ERR );
    ptr--;

    if ( *ptr != 'D' )
        return( ERR );
    ptr--;

    if ( *ptr != '.' )
        return( ERR );
    ptr--;

    return( OK );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  valid_comp_name()
*
* MODULE FUNCTION:
*
*   This routine validates the Comp filename name.  Valid Comp filenames can
*   be identified by a ".CMP" extension.  This routine was designed to be fast
*   and robust.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  valid_comp_name( name )

    char *name;

{
    char *ptr;


    if ( strlen(name) < 5 )
        return( ERR );

    ptr = &name[strlen(name)-1];

    if ( *ptr != 'P' )
        return( ERR );
    ptr--;

    if ( *ptr != 'M' )
        return( ERR );
    ptr--;

    if ( *ptr != 'C' )
        return( ERR );
    ptr--;

    if ( *ptr != '.' )
        return( ERR );
    ptr--;

    return( OK );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  write_comp_file()
*
* MODULE FUNCTION:
*
*   This routine creates a Comp file based on the supplied arguments.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  write_comp_file( compFile, lsize, purposeStr, typeList, elementList, rootElem )

    char *compFile,                 /* filename of the comp file to write   */
         elementList[][MAX_NAME],   /* array of element filenames           */
        *purposeStr[],              /* comp purpose string                  */
        *rootElem,                  /* may specify a root element name      */
         typeList[];                /* array of element types -> comp/lib   */
    int   lsize;                    /* size of the element name/type lists  */

{
    FILE *cfp;
    int   i;


    /*
     *  Write out the Comp file.
     */

    if (! (cfp = fopen(compFile,"w")) )
        {
        user_ack("Couldn't open the Comp file for writing, Comp file invalid");
        elog(1,"Couldn't open the Comp file for writing: %s", compFile);
        return;
        }

    /*
     *  Write the purpose string.
     */

    save_element_str( cfp, purposeStr );
    fprintf( cfp, "\n" );

    /*
     *  If the user has specified a new root element, make it the head of the
     *  element list and by default the new root element.
     */

    if ( rootElem )
        fprintf( cfp, "%c %s\n", 'E', rootElem );

    /*
     *  Save the element names and types.
     */

    for ( i=0; i<lsize; i++ )
        fprintf( cfp, "%c %s\n", typeList[i], elementList[i] );

    fprintf( cfp, "Z ZZ\n" );

    /*
     *  Save the symbol table.
     */

    if ( symbol_table_save(cfp) )
        {
        user_ack("Couldn't save the symbol table to the Comp file");
        elog(1,"Couldn't save the symbol table to: %s", compFile);
        user_ack("Unrecoverable error - GCB exiting");
        exit( ERR );
        }

    fclose( cfp );
}
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    comp_file.h
*
*
* FILE FUNCTION:
*
*   This file contains the function prototypes for comp_file.c
*
*
* SPECIFICATION DOCUMENTS:
*
*   /home/project/3531/Docu/GCB.spec.doc
*
*
* FILE MODULES:
*
*   N/A
*
*****************************************<----->*****************************************/


/*
 *  Function prototypes.
 */

XtCallbackProc  cbr_comp_purpose    (),
                cbr_pos_done        (),
                cbr_new_comp        (),
                cbr_sel_comp        ();

int             read_comp_list      ();

void            cbr_comp_selected   (),
                init_comp_vars      (),
                load_read_list      (),
                del_element_from_comp(),
                save_comp_file      (),
                update_comp_file    (),
                write_comp_file     ();
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    constants.h
*
*
* FILE FUNCTION:
*
*   This file contains many of the Graphical Comp Builder program constants.
*
*
* FILE MODULES:
*
*   N/A
*
*
*****************************************<----->*****************************************
/


/*
 *  Assorted constants.
 */

#define NO_WAIT                 0
#define WAIT                    1

#define HELP                    200
#define HELP_CANCEL             201
#define HELP_HELP               202
#define HELP_HELP_CANCEL        203
#define DEF_FN_CANCEL           204


/*
 *  Define Popup ids and Help keys.
 */

#define ASK_HELP                106
#define ASK_NO                  107
#define ASK_YES                 108

#define USER_ACK                143
#define USER_HELP               144


#define FILE_OK                 200
#define FILE_HELP               201

/*
 *  Help keys - these are used as indexes into the tokens array.  See tokens.h
 */

#define PALETTE_AREA            10
#define WORK_AREA               11
#define BROWSE                  12
#define ASK                     13
#define COPY_ELEM               14
#define CREATE_ELEM             15
#define SELECT_ELEM             16
#define PRINT_ELEM              17
#define TARGET_LANG             18
#define DELETE_ELEM             19
#define SET_COLORS              20
#define CREATE_COMP             21
#define SELECT_COMP             22
#define CREATE_POS              23
#define SELECT_POS              24
#define LOGIC_POPUP             25
#define CALL_ELEM               26
#define PAUSE_ELEM              27
#define TEXT_POPUP              28
#define DEF_FN_HELP             29
#define OBJECT_HELP             30
#define COMP_PURP_HELP          31
#define SYM_ATTR_HELP           32
#define VARIABLE_HELP           33
#define NUMBER_HELP             34
#define STATUS_HELP             35
#define SEL_ROOT_ELEM           36
#define DISPLAYER_HELP          37
#define USER_ACK_HELP           38
#define WSG_HELP                39
#define STRING_HELP             40
#define ACTIVATE_COMP           41
```

```
/*****************************************<----->*****************************************
*
*
* MODULE NAME:  cr_cascade()
*
* MODULE FUNCTION:
*
*   This function is an interface "helper routine" which is used to create a
*   cascade button and to place it.  This function returns a pointer to the
*   created cascade button.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/CascadeB.h>


Widget  cr_cascade( instance, parent, menu, mnemonic, label )

    Widget  menu,
            parent;
    char    *instance,
            *label,
            mnemonic;

{
    static  Arg     args[3];
            Widget  widget;
            int     n;

    n = 0;
    XtSetArg( args[0], XmNsubMenuId, menu ); n++;

/*DEBUG
    XtSetArg( args[1], XmNmnemonic,  mnemonic ); n++;
    XtSetArg( args[2], XmNfontList,  Fnt_List_Btn ); n++;
*/

    /*
     *  Use the MOTIF XmCreateCasadeButton to create the Cascade
     *  widget and attach it to the parent, and initialize all arguments.
     */

    widget = XmCreateCascadeButton( parent, label, args, n );
    XtManageChild( widget );
    return( widget );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cr_command()
*
* MODULE FUNCTION:
*
*   This function is an interface "helper routine" which is used to create a
*   push button and place it in a form.  This function returns a pointer to the
*   created widget.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include <Xm/PushB.h>
#include <Xm/Label.h>

#include "gcb.h"


Widget  cr_command( instance, parent, label, callback, id )

    XtCallbackProc  callback;
    Widget          parent;
    int             id;
    char            *instance,
                    *label;
{
    Widget          widget;
    int             n = 0;
    Arg             args[3];
    XmString        tcs;

    Dimension wid;


    /*
     *  Create compound string for the button text.
     */

    tcs = XmStringLtoRCreate( label, XmSTRING_DEFAULT_CHARSET );
    XtSetArg( args[n], XmNlabelType, XmSTRING ); n++;
    XtSetArg( args[n], XmNlabelString, tcs ); n++;

    /*
     *  Create the command button.
     */

    widget = (Widget) XmCreatePushButton( parent, instance, args, n );
    XtManageChild( widget );
    XmStringFree ( tcs );

    /*
     *  Check the width, if it is too small, reset the size.  This will ensure
     *  all the push buttons are consistent in size.
     */

    XtSetArg( args[0], XmNwidth, &wid );
    XtGetValues( widget, args, 1 );
    if ( (int) wid < MIN_BTN_WIDTH )
        {
        XtSetArg( args[0], XmNwidth, MIN_BTN_WIDTH );
        XtSetValues( widget, args, 1 );
        }

    /*
     *  Add the callback routine if specified.
     */

    if ( callback )
        XtAddCallback( widget, XmNactivateCallback, callback, id );

    return( widget );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cr_form()
*
* MODULE FUNCTION:
*
*   This function is an interface "helper routine" which is used to create a
*   form widget.  This function returns a pointer to the created widget.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/


#include <X11/Intrinsic.h>
#include <Xm/Form.h>


Widget  cr_form( instance, parent, h_offset, v_offset )

    Widget  parent,
            h_offset,
            v_offset;

    char    *instance;

{
    static  Arg     args[1];
            Widget  widget;


    /*
     *  Use the MOTIF XmCreateForm routine to create the form widget.  Then
     *  attach the newly created widget to its siblings.
     */

    XtSetArg( args[0], XmNborderWidth, 0 );

    widget = XmCreateForm( parent, instance, args, 1 );
    XtManageChild( widget );

    set_attach_widget( widget, v_offset, NULL, h_offset, NULL );

    return( widget );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cr_frame()
*
* MODULE FUNCTION:
*
*   This function is an interface "helper routine" which is used to create a
*   frame widget.  This function returns a pointer to the created widget.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

#include <X11/Intrinsic.h>
#include <Xm/Frame.h>


Widget  cr_frame( instance, parent, h_offset, v_offset )

    Widget  parent,
            h_offset,
            v_offset;
    char    *instance;

{
    Widget  widget;

    /*
     *  Use the MOTIF XmCreateFrame routine to create the frame widget.  Then
     *  attach the newly created widget to its siblings.
     */

    widget = XmCreateFrame( parent, instance, NULL, 0 );
    XtManageChild( widget );

    set_attach_widget( widget, v_offset, NULL, h_offset, NULL );

    return( widget );
}
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    cr_label.c
*
* FILE FUNCTION:
*
*   This file contains the routines which create and manipulate label widgets.
*
*
* FILE MODULES:
*
*   cr_label()  - create and manage a label widget
*   set_label() - set the label widget's text string to the specified string
*
*****************************************<----->*****************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/Label.h>

#include "gcb.h"


/*****************************************<----->*****************************************
*
* MODULE NAME:  cr_label()
*
* MODULE FUNCTION:
*
*   This routine is used to build label widgets which are placed within forms.
*   This routine returns a pointer to the newly created and managed widget.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

Widget  cr_label( instance, parent, label, borderWidth, top, bottom, left, right )

    Widget  parent;
    char    *instance,
            *label;
    int     borderWidth,
            top, bottom, left, right;

{
    Widget          widget;
    register int    n = 0;
    Arg             args[5];
    XmString        tcs;

    /*
     *  Create compound string for the label text.
     */

    tcs = XmStringLtoRCreate( label,   XmSTRING_DEFAULT_CHARSET );
    XtSetArg( args[n], XmNlabelType,   XmSTRING );     n++;
    XtSetArg( args[n], XmNlabelString, tcs );          n++;
    XtSetArg( args[n], XmNborderWidth, borderWidth );  n++;

    /*
     *  Set the font of the label string.
     */

/*DEBUG
    XmFontList      fnt_list;

    if (load_font("9x15", &fnt_list))
        {
        user_ack("cr_text: loadfont failed");
        exit(1);
        }

    XtSetArg( args[n], XmNfontList,  fnt_list );  n++;
*/

    /*
     *  Create the label widget.
     */

    widget = (Widget) XmCreateLabel( parent, instance, args, n );
    XtManageChild( widget );
    XmStringFree ( tcs );

    /*
     *  Place the new widget relatively within the form.
     */

    set_position( widget, top, bottom, left, right );

    return( widget );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  set_label()
*
* MODULE FUNCTION:
*
*   This routine sets a label widget to the specified string value.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  set_label( widget, string )

    Widget  widget;
    char    *string;

{
    Arg args[1];

    XtSetArg( args[0], XmNlabelString,
              XmStringCreate(string,XmSTRING_DEFAULT_CHARSET) );
    XtSetValues( widget, args, 1 );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cr_list()
*
* MODULE FUNCTION:
*
*   This function is an interface "helper routine" which is used to create a
*   scrolled list widget.  This function returns a pointer to the created
*   widget.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/


#include <X11/Intrinsic.h>
#include <Xm/List.h>


Widget  cr_list( instance, parent, count, width )

    Widget  parent;
    char    *instance;
    int     count,
            width;

{
    static  Arg     args[4];
            Widget  widget;

    XtSetArg( args[0], XmNvisibleItemCount,         count );
    XtSetArg( args[1], XmNwidth,                    width );
    XtSetArg( args[2], XmNlistSizePolicy,           XmCONSTANT );
    XtSetArg( args[3], XmNscrollBarDisplayPolicy,   XmSTATIC );

    /*
     *  Use the MOTIF XmCreateScrolledList routine to create the list
     *  widget, attach it to the parent, and initialize all arguments.
     */

    widget = XmCreateScrolledList( parent, instance, args, 4 );
    XtManageChild( widget );

    return( widget );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cr_pixmap()
*
* MODULE FUNCTION:
*
*   This routine is used to build pixmap label widgets which are placed within
*   forms.  A pointer to the newly created and managed widget is returned.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/


#include <X11/Intrinsic.h>
#include <Xm/Label.h>
#include "gcb.h"


Widget  cr_pixmap( instance, parent, pixmap, top, bottom, left, right )


    Pixmap  *pixmap;
    Widget  parent;
    char    *instance;
    int     top,  bottom, left, right;

{
    Widget  widget;
    Arg     args[2];

    XtSetArg( args[0], XmNlabelType,   XmPIXMAP );
    XtSetArg( args[1], XmNlabelPixmap, *pixmap );

    /*
     *  Create the label widget with a pixmap.
     */

    widget = (Widget) XmCreateLabel( parent, instance, args, 2 );
    XtManageChild( widget );

    /*
     *  Place the new widget relatively within the form.
     */

    set_position( widget, top, bottom, left, right );

    return( widget );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cr_popup()
*
* MODULE FUNCTION:
*
*   This function is an interface "helper routine" which is used to create a
*   popup shell.  This function returns a pointer to the created widget.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/


#include <X11/Intrinsic.h>
#include <Xm/MwmUtil.h>
#include <Xm/BulletinB.h>


Widget  cr_popup( instance, parent, title )

    Widget  parent;
    char    *instance,
            *title;

{
    Widget      widget, foo;
    Arg         args[2];
    XmString    xstr;
    int         n = 0;


    /*
     *  If a title string was supplied for the popup, initialize an argument.
     */

    if ( title )
        {
        xstr = XmStringCreate( title, XmSTRING_DEFAULT_CHARSET );
        XtSetArg( args[n], XmNdialogTitle, xstr ); n++;
        }

    /*
     *  Set the dialog style to MODAL, this will ensure that the user
     *  processes the popup before doing anything else within the GCB.
     */

    XtSetArg( args[n], XmNdialogStyle, XmDIALOG_APPLICATION_MODAL ); n++;


    XtSetArg( args[n], XmNmwmDecorations, 0 ); n++;

    /*
     *  Create the popup widget.
     */

    widget = XmCreateBulletinBoardDialog( parent, instance, args, n );

    /*
     *  Remove the window menu button and the close function from
     *  the popup.
     */

    XtSetArg( args[0], XmNmwmDecorations, 17 ); n++;
    XtSetArg( args[1], XmNmwmFunctions, 33 ); n++;
    XtSetValues( XtParent(widget), args, 2 );

    /*
     *  Free the string now that we are finished with it.
     */

    XmStringFree ( xstr );

    return( widget );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cr_pulldown()
*
* MODULE FUNCTION:
*
*   This function is an interface "helper routine" which is used to create a
*   pulldown menu.  This function returns a pointer to the created widget.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/RowColumn.h>


Widget  cr_pulldown( instance, parent )

    Widget          parent;
    char            *instance;

{
    /*
     *  Create the pulldown menu.
     */

    return( (Widget) XmCreatePulldownMenu(parent,instance,NULL,0) );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cr_radio_box()
*
* MODULE FUNCTION:
*
*   This function is an interface "helper routine" which is used to create a
*   radio box and place it in a form.  This function returns a pointer to the
*   created widget.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include "gcb.h"


Widget  cr_radio_box( instance, parent, orientation )

    Widget          parent;
    char            *instance;
    unsigned char   orientation;

{
    Widget          widget;
    Arg             args[1];


    /*
     *  Create the radio box.
     */

    XtSetArg( args[0], XmNorientation, orientation );

    widget = (Widget) XmCreateRadioBox( parent, instance, args, 1 );
    XtManageChild( widget );

    return( widget );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cr_rel_cmd()
*
* MODULE FUNCTION:
*
*   This routine is used to build command widgets which are placed relatively within
*   forms.  A pointer to the newly created and managed widget is returned.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/


#include <X11/Intrinsic.h>
#include <Xm/PushB.h>
#include "gcb.h"


Widget  cr_rel_cmd( instance, parent, label, callback, id, top, left )

    Widget          parent;
    XtCallbackList  callback;
    char            *instance,
                    *label;
    int             id,
                    left, top;

{
    Widget          widget;
    register int    n = 0;
    Arg             args[6];
    XmString        tcs;


    /*
     *  Create compound string for the button text.
     */

    tcs = XmStringCreate( label, XmSTRING_DEFAULT_CHARSET );
    XtSetArg( args[n], XmNlabelType,   XmSTRING ); n++;
    XtSetArg( args[n], XmNlabelString, tcs ); n++;

    /*
     *  Create the command button.
     */

    widget = (Widget) XmCreatePushButton( parent, instance, args, n );
    XtManageChild( widget );

    XmStringFree ( tcs );

    /*
     *  Place the new widget relatively within the form.
     */

    set_position( widget, top, IGNORE, left, IGNORE );

    /*
     *  Add the callback routine if specified.
     */

    if ( callback )
        {
        callback->closure = (caddr_t) id;
        XtAddCallbacks( widget, XmNactivateCallback, callback );
        }

    return( widget );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  Ncr_rel_cmd()
*
* MODULE FUNCTION:
*
*   This routine is used to create command buttons.  This routine differs from
*   cr_rel_cmd() in that its callback parameter data type is a function instead
*   of a list of functions.  This routine should replace cr_rel_cmd() once all
*   XtCallbackLists are replaced with XtCallbackProcs.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

Widget  Ncr_rel_cmd( instance, parent, label, callback, id, top, left )

    Widget          parent;
    XtCallbackList  callback;
    char            *instance,
                    *label;
    int             id,
                    left, top;

{
    Widget          widget;
    register int    n = 0;
    Arg             args[6];
    Dimension       wid;
    XmString        tcs;


    /*
     *  Create compound string for the button text.
     */

    tcs = XmStringCreate( label, XmSTRING_DEFAULT_CHARSET );
    XtSetArg( args[n], XmNlabelType,   XmSTRING ); n++;
    XtSetArg( args[n], XmNlabelString, tcs ); n++;

    /*
     *  Create the command button.
     */

    widget = (Widget) XmCreatePushButton( parent, instance, args, n );
    XtManageChild( widget );

    XmStringFree ( tcs );

    /*
     *  Check the width, if it is too small, reset the size.  This will make
     *  sure all the buttons are of reasonable size.
     */

    XtSetArg( args[0], XmNwidth, &wid );
    XtGetValues( widget, args, 1 );

    if ( (int) wid < MIN_BTN_WIDTH )
        {
        XtSetArg( args[0], XmNwidth, MIN_BTN_WIDTH );
        XtSetValues( widget, args, 1 );
        }

    /*
    /*
     *  Place the new widget relatively within the form.
     */

    set_position( widget, top, IGNORE, left, IGNORE );

    /*
     *  Add the callback routine if specified.
     */

    if ( callback )
        XtAddCallback( widget, XmNactivateCallback, callback, id );

    return( widget );
}
```

```
/****************************************<----->****************************************
*
* MODULE NAME:  cr_rowcol()
*
* MODULE FUNCTION:
*
*   This function is an interface "helper routine" which is used to create a
*   row column widget and place it in a form.  This function returns a pointer to the
*   created widget.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
****************************************<----->****************************************/


#include <X11/Intrinsic.h>
#include <Xm/RowColumn.h>


Widget  cr_rowcol( instance, parent, columns, orientation, v_offset, h_offset )

    Widget  parent,
            v_offset, h_offset;
    char    *instance;
    int     columns,
            orientation;

{
    static  Arg     args[5];
            Widget  widget;


    XtSetArg( args[0], XmNpacking,        XmPACK_COLUMN );
    XtSetArg( args[1], XmNnumColumns,     columns );
    XtSetArg( args[2], XmNentryAlignment, XmALIGNMENT_CENTER );
    XtSetArg( args[3], XmNorientation,    orientation );

    /*
     *  Use the MOTIF XmCreateRowColumn routine to create the RowColumn widget,
     *  attach it to the parent, and initialize all arguments.
     */

    widget = XmCreateRowColumn( parent, instance, args, 4 );
    XtManageChild( widget );

    set_attach_widget( widget, v_offset, NULL, h_offset, NULL );

    return( widget );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cr_scr_text()
*
* MODULE FUNCTION:
*
*   This function is an interface "helper routine" which is used to create a scrolled
*   text widget and place it in a form.  This function returns a pointer to the
*   created widget.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/Text.h>


Widget  cr_scr_text( instance, parent, rows, edit, width, x, y )

    Widget          parent;
    char            *instance;
    int             edit,
                    rows,
                    width,
                    x, y;

{
    Widget          widget;
    register int    n = 0;
    Arg             args[8];


    /*
     *  Set the position of the widget if coordinates are supplied.
     */

    if ( x )
        {
        XtSetArg( args[n], XmNx, x );
        n++;
        }

    if ( y )
        {
        XtSetArg( args[n], XmNy, y );
        n++;
        }

    /*
     *  Initialize other attributes.
     */

    XtSetArg( args[n], XmNrows,             rows  ); n++;
    XtSetArg( args[n], XmNeditable,         edit  ); n++;
    XtSetArg( args[n], XmNwidth,            width ); n++;
    XtSetArg( args[n], XmNscrollVertical,   True  ); n++;
    XtSetArg( args[n], XmNeditMode,         XmMULTI_LINE_EDIT ); n++;

/*DEBUG
    XtSetArg( args[n], XmNscrollingPolicy,  XmAUTOMATIC  ); n++;
*/

    /*
     *  Create the widget and manage the new widget.
     */

    widget = XmCreateScrolledText( parent, instance, args, n );
    XtManageChild( widget );

    return( widget );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cr_separator()
*
* MODULE FUNCTION:
*
*   This function is an interface "helper routine" which is used to create a
*   separator widget and place it in a form.  This function returns a pointer
*   to the created widget.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/


#include <X11/Intrinsic.h>
#include <Xm/Separator.h>
#include "gcb.h"


Widget  cr_separator( instance, parent, top, bottom, left, right )

    Widget  parent;
    char    *instance;
    int     top, bottom,
            left, right;

{
    Widget  widget;

    /*
     *  Create the separator widget.
     */

    widget = XmCreateSeparator( parent, instance, NULL, 0 );
    XtManageChild( widget );

    /*
     *  Place the new widget relatively within the form.
     */

    set_position( widget, top, bottom, left, right );
    return( widget );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cr_text()
*
* MODULE FUNCTION:
*
*   This function is an interface "helper routine" which is used to create a
*   text widget and place it in a form.  This function returns a pointer
*   to the created widget.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/


#include <X11/Intrinsic.h>
#include <Xm/Text.h>


Widget  cr_text( instance, parent, h_offset, v_offset, text, scrolled, rows, columns )

    Widget  parent,
            h_offset,
            v_offset;
    char    *instance,
            *text;
    int     columns,
            rows,
            scrolled;

{
    Widget      widget;
    Arg         args[10];
    int         n = 0;


    /*
     *  Initialize the attribute array for the text widget.
     */

    XtSetArg( args[n], XmNvalue,                text );  n++;
    XtSetArg( args[n], XmNrows,                 rows );  n++;
    XtSetArg( args[n], XmNcolumns,              columns );  n++;
    XtSetArg( args[n], XmNhighlightThickness,   0 );  n++;
    XtSetArg( args[n], XmNmarginWidth,          5 );  n++;
    XtSetArg( args[n], XmNmarginHeight,         3 );  n++;

    if ( rows > 1 )
        {
        XtSetArg( args[n], XmNeditMode, XmMULTI_LINE_EDIT ); n++;
        }

    /*
     *  Load the font and then set the font of the text widget.
     */

/*DEBUG
    XmFontList  fnt_list;
    if (load_font("9x15", &fnt_list))
        {
        user_ack("cr_text: loadfont failed");
        exit(1);
        }
    XtSetArg( args[n], XmNfontList,  fnt_list );  n++;
*/

    /*
     *  Create a scrolled window with a text edit widget.
     */

    if ( scrolled )
        {
        widget = XmCreateScrolledText( parent, instance, args, n );
        XtManageChild( widget );

        set_attach_widget( XtParent(widget), v_offset, NULL, h_offset, NULL );
        }

    /*
     *  Create a non-scrolled text edit widget.
     */

    else
        {
        widget = XmCreateText( parent, instance, args, n );
        XtManageChild( widget );

        set_attach_widget( widget, v_offset, NULL, h_offset, NULL );
        }

    return( widget );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cr_toggle()
*
* MODULE FUNCTION:
*
*   This function is an interface "helper routine" which is used to create a toggle
*   button and place it in a form.  This function returns a pointer to the created
*   widget.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/


#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/ToggleB.h>
#include <Xm/Xm.h>


Widget  cr_toggle( instance, parent, label, callback, arm, disarm )

    Widget          parent;
    XtCallbackList  callback;
    char            *instance,
                    *label;
    int             arm,
                    disarm;

{
    Arg             args[5];
    Widget          widget;
    XmString        tcs;
    register int    n = 0;


    /*
     *  Create compound string for the button text.
     */

    tcs = XmStringCreate( label, XmSTRING_DEFAULT_CHARSET );

    /*
     *  Initialize attribute array.
     */

    XtSetArg( args[n], XmNlabelType,            XmSTRING ); n++;
    XtSetArg( args[n], XmNlabelString,          tcs ); n++;
    XtSetArg( args[n], XmNhighlightThickness,   0   ); n++;

/*DEBUG
    XtSetArg( args[n], XmNfontList, Fnt_List_Btn ); n++;
*/

    /*
     *  Use the MOTIF XmCreateToggleButton routine to create the widget.  Then
     *  manage the widget and free() the compound string.
     */

    widget = XmCreateToggleButton( parent, instance, args, n );
    XtManageChild( widget );

    XmStringFree ( tcs );

    /*
     *  Add the callback routine if specified.
     */

    if ( callback )
        {
        callback->closure = (caddr_t) arm;
        XtAddCallbacks( widget, XmNarmCallback, callback );

        callback->closure = (caddr_t) disarm;
        XtAddCallbacks( widget, XmNdisarmCallback, callback );
        }

    return( widget );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  Ncr_toggle()
*
* MODULE FUNCTION:
*
*   This function is an interface "helper routine" which is used to create a toggle
*   button and place it in a form.  This function returns a pointer to the created
*   widget.  This function should replace cr_toggle() once all XtCallbackLists are
*   removed.  This function accepts XtCallbackProcs whereas cr_toggle() accepts
*   XtCallbackLists.  This routine will replace cr_toggle() once all the
*   XtCallbackLists are replaced with XtCallbackProcs.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

Widget  Ncr_toggle( instance, parent, label, callback, arm, disarm )

    Widget          parent;
    XtCallbackProc  callback;
    char            *instance,
                    *label;
    caddr_t         arm,
                    disarm;

{
    Arg             args[5];
    Widget          widget;
    XmString        tcs;
    register int    n = 0;


    /*
     *  Create compound string for the button text.
     */

    tcs = XmStringCreate( label, XmSTRING_DEFAULT_CHARSET );

    /*
     *  Initialize attribute array.
     */

    XtSetArg( args[n], XmNlabelType,            XmSTRING ); n++;
    XtSetArg( args[n], XmNlabelString,          tcs ); n++;
    XtSetArg( args[n], XmNhighlightThickness,   0   ); n++;

/*DEBUG
    XtSetArg( args[n], XmNfontList, Fnt_List_Btn ); n++;
*/

    /*
     *  Use the MOTIF XmCreateToggleButton routine to create the widget.  Then
     *  manage the widget and free() the compound string.
     */

    widget = XmCreateToggleButton( parent, instance, args, n );
    XtManageChild( widget );

    XmStringFree ( tcs );

    /*
     *  Add the callback routine if specified.
     */

    if ( callback )
        {
        XtAddCallback( widget, XmNarmCallback,    callback, arm );
        XtAddCallback( widget, XmNdisarmCallback, callback, disarm );
        }

    return( widget );
}
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    cursors.h
*
*
* FILE FUNCTION:
*
*   This file contains the cursor names used throughout the GCB
*
*
* SPECIFICATION DOCUMENTS:
*
*   /home/project/3531/Docu/GCB.spec.doc
*
*
* FILE MODULES:
*
*   N/A
*
*****************************************<----->*****************************************/

Cursor  tcross_cursor, ul_cursor, lr_cursor, basic_cursor, clock_cursor;
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    data_type.c
*
* FILE FUNCTION:
*
*   This file contains the routines which perform analysis of an expression to determine
*   is "compilability" -- that is, not whether or not the expression has valid syntax
*   (YACC supplies this) but whether or not the expression will pass through the
*   compiler (i.e. a floating point number should not be assigned to an integer).
*   Any expression passed to these routines will be syntactically correct, these
*   routines are looking for any data type anamolies.
*
*   For the purposes of evaluation, data types are considered in the following order:
*
*           DOUBLE
*           FLOAT
*           UNSIGNED
*           INTEGER
*           SHORT
*           CHAR
*
*   It is assumed that variables of differing types can be assigned to a more dominant
*   type without loss of data.
*
*   Te GCB code builder will take the expression provided by the user and place it in
*   the C source code file for compilation (it is assumed that the statement is sytact-
*   ically correct through the use of YACC).
*
*   In addition, the GCB allows the user to enter inline expressions with matrix
*   variables embedded, while this is valid syntax (according to the GCB BNF), it will
*   not produce working compiled code.  Therefore, the routines in this file will
*   provide matrix information so that the expression supplied by the user can be
*   mapped to matrix manipulation routines provided in the "skeleton_element.c" source
*   file.
*
*   The GCB invokes the routines in this file to return informative diagnostics about
*   the data types contained in the supplied expression.  The GCB will not allow a user
*   to enter expressions that will cause compile and run time anamolies.  The following
*   routines will access a variety of global (statically allocated) variables which are
*   use to correctly and fully interpret the characteristics of a particular expression.
*
*
* FILE MODULES:
*
*   set_bracket()               - issues a state change to acknowledge that either right
*                                 or left bracket ([ or ]) was encountered
*   func_state()                - issues a state change to indicate that a function's
*                                 actual parameters are being processed
*   set_func_type()             - sets the data type of the function being invoked
*   save_type()                 - saves the type of a scalar variable
*   process_id_data_type()      - saves the type (and attributes if a matrix) of an
*                                 indentifier
*   save_operator()             - saves an operator found in an expression
*   init_matrix_structure()     - initializes a blank matrix data structure
*   verify_expression_type()    - entry point from the main COMP builder which is used
*                                 to determine if an expression has a legitimate mix of
*                                 data types
*   return_matrix_data()        - returns specific matrix data when a matrix operation
*                                 is required
*   process_matrix_dimensions   - verfiles that the number of subscripts supplied by the
*                                 user is consistent with the matrix definition
*   matrix_in_expr()            - determines if a matrix is in an expression
*
*****************************************<----->*****************************************/

/*
 *  COTS include files.
 */

#include <stdio.h>
#include <memory.h>
#include <X11/Intrinsic.h>
#include <X11/Shell.h>
#include <Xm/Xm.h>

/*
 *  Custom include files.
 */

#include "gcb.h"
#include "symbol.h"
#include "gcb_parse.h"

/*
 *  Various local defines.
 */

#define UNDEFINED_OPERATION     0
#define GLOBAL_ERROR_SIZE       128

/*
 *  External references to parsing data structures.
 */

extern int WhereAmI;

/*
 *  Global variables which will hold the current data type for the LHS and RHS of
 *  the expression.  These variables are only utilized while a SINGLE expression is
 *  being evaluated.
 */

static int parse_status, function_status, bracket_count, number_dimensions;
static int lhs_current_attributes, rhs_current_attributes, number_operators;
static int last_operator, matrix_found;
static struct matrix_data result, operand_1, operand_2;
static char current_matrix_id[ MAX_SYMBOL_SIZE + 1 ], global_error[ GLOBAL_ERROR_SIZE + 1
 ];
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  set_bracket()
*
*
* MODULE FUNCTION:
*
*  Function set_bracket is invoked inform the expression analyzer that a bracket was
*  encounter -- this is important because the analyzer will assume that an operation
*  is matrix in nature when the identifier is found to be of type MATRIX; this function
*  is invoked because YACC has found a matrix element access (i.e. x[1][2]) and there-
*  fore the identified access should be considered to be a scalar operation.  This
*  function is only invoked when a left bracket is found (the YACC grammar will ensure
*  that a companion right bracket exists).
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

void set_bracket()
{
    /*
     *  This routine is also used to determine if an identifier (which was declared as
     *  a matrix) is accessed properly if subelements are extracted.
     */

    bracket_count++;

    /*
     *  If we had previously assume a matrix operation (based on the identifier type,
     *  unmark the type so that correct interpretations are made (i.e. do not confuse
     *  the matrix "abc" with "abc[1][2]", the latter should be considered as a scalar
     *  operation).  If we are on the RHS and brackets were encountered, turn off the
     *  appropriate bit mask of the operand (based on the operator count).
     */

    if ( WhereAmI == LHS )
        lhs_current_attributes &= ~MATRIX;
    else {
        rhs_current_attributes &= ~MATRIX;

        /*
         *  If we are on operand 1 or two, appropriately mask off bits.
         */

        switch( number_operators ) {
            case 0  : operand_1.md_attributes &= ~MATRIX;
                      break;
            case 1  : operand_2.md_attributes &= ~MATRIX;
                      break;
            default : /*
                       *  Do nothing.
                       */

                      break;
        }
    }

    /*
     *  The other use of this function is to determine if a matrix appears in a
     *  conditional expression.  This function will decriment the matrix_found
     *  count by one each time it is encountered (each matrix element access will
     *  invoke this function twice).
     */

    matrix_found--;
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  func_state()
*
*
* MODULE FUNCTION:
*
*  Function func_status is invoked when a function call is started.  When parameters
*  appear within a set of perenthesis then the data type is ignored.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

void func_state( status )

    int status;

{
    /*
     *  If the status is turned to ON, increment the perenthesis count for function
     *  calls; otherwise, decrement the count.
     */

    if ( status == ON )
        function_status++;
    else
        function_status--;
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  set_func_type()
*
*
* MODULE FUNCTION:
*
*  The function func_state is invoked to save the data type of a function being invoked;
*  all intrinsic routines are of type DOUBLE (UNIX conventions) and all user defined
*  functions are of type INTEGER (GCB convention).
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

void set_func_type( function_type )

    int function_type;

{
    if ( function_type == USER_DEFINED )
        rhs_current_attributes |= INTEGER;
    else
        rhs_current_attributes |= DOUBLE;
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  save_type()
*
*
* MODULE FUNCTION:
*
*  Function save_type is invoked to save the type of a scaler value found by YACC (i.e.
*  this function is invoked when non-variables are located).  The function will update
*  the RHS current attributes variable based on the data type found by YACC.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

void save_type( data_type )

    int data_type;

{
    /*
     *  If a function call is currently open (determined by function_status > 0) then
     *  ignore the type save because the type should not be used in determining data
     *  type.
     */

    if ( function_status > 0 )
        return;

    /*
     *  This function should only be invoked when the RHS is being evaluated; if
     *  not, an error needs to be generated.  Otherwise, OR in the data type to the
     *  RHS current attributes variable.
     */

    if ( WhereAmI == LHS )
        parse_status = ERR;
    else
        rhs_current_attributes |= data_type;
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  process_matrix_dimensions()
*
*
* MODULE FUNCTION:
*
*  Function process_matrix_dimensions will check the number of subscripts performed on
*  on a matrix ID.  If the number is incorrect appropriate errors will be generated.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

void process_matrix_dimensions()
{
    /*
     *  If the array was referenced with brackets, verify that the bracket count equals
     *  the number of dimensions specified; if not, generate a diagnostic.
     */

    if ( bracket_count )
        if ( number_dimensions != bracket_count ) {
            if ( bracket_count < number_dimensions )
                sprintf( global_error,
                        "matrix \"%s\" is a %dD array, referenced with only %d dimension(
s)\n",
                        current_matrix_id,
                        number_dimensions,
                        bracket_count );
            else
                sprintf( global_error,
                        "matrix \"%s\" is a %dD array, referenced with %d dimension(s)\n"
,
                        current_matrix_id,
                        number_dimensions,
                        bracket_count );
            parse_status = ERR;
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  process_id_data_type()
*
*
* MODULE FUNCTION:
*
*  Function process_id_data_type is invoked to process the attributes of an identifier
*  located by YACC.  If the identifier located is a matrix various attribute information
*  is stored so that appropriate data type checks can be made at the conclusion of the
*  expression analysis.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

void process_id_data_type( current_symbol )

    char *current_symbol;

{
    register int attributes;
    struct matrix_data *temp_matrix_data;
    short dims, sub1, sub2, sub3, sub4;

    /*
     *  If a function call is currently open (determined by function_status > 0) then
     *  ignore the type save because the type should not be used in determining data
     *  type.
     */

    if ( function_status > 0 )
        return;

    /*
     *  Determine the attributes of the symbol.  First attempt to resolve the variable
     *  from the current element, if unresolved, look in the global symbol table.
     */

    if ( ( attributes = return_symbol_attributes(ElementFile,current_symbol) ) == ERR )
        if ( ( attributes = return_symbol_attributes( NULL, current_symbol ) ) == ERR )
            parse_status = ERR;

    /*
     *  Determine if the operation is a MATRIX operation.
     */

    if ( attributes & MATRIX ) {

        /*
         *  Update parse status.  Additionally, update the matrix_found variable to
         *  indicate that a matrix identifier was located.  The variable is incremented
         *  by two because each matrix element access must encounter two left brackets
         *  (which will cause the variable to be decremented by one each time).
         */

        parse_status = MATRIX_OPERATION;
        matrix_found += 2;

        /*
         *  At this point of the check, it is unknown whether or not the user will be
         *  accessing an element of the array or the entire array.  The bracket_count
         *  variable is initialize to zero to check for this condition.  Save the matrix
         *  name and the number of dimensions for which is was declared (which will be
         *  used later during diagnostics).
         */

        bracket_count = 0;
        strcpy( current_matrix_id, current_symbol );
        dims = 0;
        if ( return_matrix_attributes( ElementFile, current_symbol,
                                       &dims, &sub1, &sub2, &sub3, &sub4 ) == ERR )
            if ( return_matrix_attributes( NULL, current_symbol,
                                           &dims, &sub1, &sub2, &sub3, &sub4 ) == ERR )
                parse_status = ERR;
        number_dimensions = dims;
    }
    else
        parse_status = SCALAR_OPERATION;

    /*
     *  Set the appropriate variables based on whether or not we are parsing the left or
     *  right hand side.
     */

    switch( WhereAmI ) {
        case LHS : /*
                    *  Save the attributes and the name.
                    */

                   lhs_current_attributes |= attributes;
                   strcpy( result.md_name, current_symbol );

                   /*
                    *  If the operation is a MATRIX operation, then the number of rows
                    *  and columns of the matrix must be retrieved.
                    */

                   if ( parse_status == MATRIX_OPERATION ) {

                       /*
                        *  Save the attributes.
                        */

                       result.md_attributes = attributes;

                       /*
                        *  Go and fetch the number of rows and columns from the symbol
                        *  table. This information will be used for integrity checking
                        *  of expressions.
                        */

                       if ( return_matrix_attributes( ElementFile,
                                                      current_symbol,
                                                      &result.md_num_dimensions,
                                                      &result.md_subs[ SUB1 ],
                                                      &result.md_subs[ SUB2 ],
                                                      &result.md_subs[ SUB3 ],
                                                      &result.md_subs[ SUB4 ] ) == ERR)
                           if ( return_matrix_attributes( NULL,
                                                          current_symbol,
                                                          &result.md_num_dimensions,
```

```
                                        &result.md_subs[ SUB1 ],
                                        &result.md_subs[ SUB2 ],
                                        &result.md_subs[ SUB3 ],
                                        &result.md_subs[ SUB4 ] )
                                         == ERR )
                    parse_status = ERR;
            }
            break;
    case RHS : rhs_current_attributes |= attributes;

            /*
             *  If the operation is a MATRIX operation, then the number of rows
             *  and columns of the matrix must be retrieved.
             */

            if ( parse_status == MATRIX_OPERATION ) {

                /*
                 *  Establish temporary pointers to local data structures.
                 */

                if ( number_operators == 0 )
                    temp_matrix_data = &operand_1;
                else
                    temp_matrix_data = &operand_2;

                /*
                 *  Save the name and attributes.
                 */

                strcpy( temp_matrix_data->md_name, current_symbol );
                temp_matrix_data->md_attributes = attributes;

                /*
                 *  Go and fetch the number of rows and columns from the symbol
                 *  table. This information will be used for integrity checking
                 *  of expressions.
                 */

                if ( return_matrix_attributes(
                                ElementFile,
                                current_symbol,
                                &temp_matrix_data->md_num_dimensions,
                                &temp_matrix_data->md_subs[ SUB1 ],
                                &temp_matrix_data->md_subs[ SUB2 ],
                                &temp_matrix_data->md_subs[ SUB3 ],
                                &temp_matrix_data->md_subs[ SUB4 ] ) == ERR )
                    if ( return_matrix_attributes(
                                NULL,
                                current_symbol,
                                &temp_matrix_data->md_num_dimensions,
                                &temp_matrix_data->md_subs[ SUB1 ],
                                &temp_matrix_data->md_subs[ SUB2 ],
                                &temp_matrix_data->md_subs[ SUB3 ],
                                &temp_matrix_data->md_subs[ SUB4 ] ) == ERR )
                        parse_status = ERR;
            }
            break;
    }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  save_operator()
*
*
* MODULE FUNCTION:
*
*  Function save_operator is invoked when YACC locates an operator.  The operator
*  count is updated and the last operator encountered is recorded.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

void save_operator( operator )

    int operator;

{
    number_operators++;
    last_operator = operator;
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  dominant_type()
*
*
* MODULE FUNCTION:
*
*  Function dominant_type will return the most dominant type for the RHS of an expressio
n.
*  The most dominant type is defined as the data type which requires highest number of
*  bits to represent (i.e. type DOUBLE is considered more dominant than data type INTEGE
R).
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

static int dominant_type( data_type )

    int data_type;

{
    register int mask;

    /*
     *  Look for the leftmost bit of the data_type.  When found, return the bit.
     */

    for ( mask = DOUBLE; mask != 0; mask >>= 1 )
        if ( mask & data_type )
            return( mask );

    /*
     *  No data type was found, return a zero (0).
     *  found.
     */

    return( 0 );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  init_matrix_structure()
*
*
* MODULE FUNCTION:
*
*  Function init_matrix_structure is invoked to clear out a matrix data structure --
*  these data structures are used during the evaluation of an expression to determine
*  if the expression specified is "compilable".
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

void init_matrix_structure( structure )

    struct matrix_data *structure;

{
    /*
     *  Initialize the various fields of the matrix data structure.
     */

    structure->md_name[ 0 ] = NULL;
    structure->md_attributes = 0;
    structure->md_subs[ SUB1 ] = 0;
    structure->md_subs[ SUB2 ] = 0;
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  verify_expression_type()
*
*
* MODULE FUNCTION:
*
*  Function verify_expression_type is invoked to determine if a particular expression
*  will produce reliable source code.  The function will initialize various global stati
c
*  variables to record the specific information about an expression.  The routine
*  parse_expression is invoked to have YACC invoke the various routines which will recor
d
*  data type information.  The function will return a diagnostic and explanatory char-
*  acter message to the invoking routine in the event a condition with a bad mix of data
*  types is encountered.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

int verify_expression_type( expression, response_message )

    char *expression, *response_message;

{
    int i;
    /*
     *  Initialize various control variables.
     */

    parse_status = UNDEFINED_OPERATION;
    function_status = 0;
    last_operator = 0;
    bracket_count = 0;
    response_message[ 0 ] = NULL;
    global_error[ 0 ] = NULL;
    lhs_current_attributes = 0;
    rhs_current_attributes = 0;
    number_operators = 0;
    init_matrix_structure( &result );
    init_matrix_structure( &operand_1 );
    init_matrix_structure( &operand_2 );

    /*
     *  Invoke the parse_expression routine to cause YACC to analyze the expression
     *  supplied.  YACC will cause the previously defined functions to be invoked in
     *  order to record data type information.
     */

    i = parse_expression( expression );
    if ( i == END_OF_FILE )
        return( VALID_EXPRESSION );
    else if ( i != SUCCESS )
        {
        strcpy( response_message, "Syntax error in expression" );
        return( INVALID_EXPRESSION );
        }

    /*
     *  Reset the value of parse expression (if not in an error condition), if all
     *  attributes are scalar in nature, set the variable appropriately, otherwise
     *  the operation is assume to be a matrix.
     */

    if ( parse_status == ERR ) {
        if ( global_error[ 0 ] != NULL )
            strcpy( response_message, global_error );
        else
            strcpy( response_message, "Syntax error in expression" );
        return( INVALID_EXPRESSION );
    }

    /*
     *  Assume that the expression consists of scalar operations unless the MATRIX
     *  attributes were set during the analysis cycle.
     */

    parse_status = SCALAR_OPERATION;
    if ( ( lhs_current_attributes & MATRIX ) || ( rhs_current_attributes & MATRIX ) )
        parse_status = MATRIX_OPERATION;

    /*
     *  If the operation is a SCALAR operation then verify that the left and right
     *  side variable types are compatible.  If not, generated an error.
     */

    if ( parse_status == SCALAR_OPERATION ) {

        /*
         *  Verify that the dominant RHS data type (the data type with the highest
         *  precision) is of lesser or equal precision than the LHS/
         */

        if ( dominant_type( rhs_current_attributes ) > lhs_current_attributes ) {
            strcpy( response_message,
                    "Assignment of RHS to LHS would cause data loss" );
            return( INVALID_EXPRESSION );
        }
        else

            /*
             *  If any of the data types are type character, make sure that the
             *  expression is only a simple assignment (no operators).
             */
            if ( (lhs_current_attributes & CHAR) || (rhs_current_attributes & CHAR) ) {

                /*
                 *  Since the data type CHAR has a lower value than the data type INT,
                 *  make sure that INT and CHAR are not mixed on an assignment line.
                 */

                if ( ( lhs_current_attributes & CHAR ) == 0 ) {
                    strcpy( response_message, "LHS must be of data type STRING" );
                    return( INVALID_EXPRESSION );
                }

                /*
                 *  Make sure that the RHS ONLY contains a single character variable
                 *  (i.e. no operators).
                 */
```

```
            if ( number_operators != 0 ) {
                strcpy( response_message,
                        "Cannot mix operators and string variables" );
                return( INVALID_EXPRESSION );
            }
        }

        /*
         *  Make sure that no martrix operators were used in the scalar
         *  operation.
         */

        if ( ( last_operator == MADD_OPER ) ||
             ( last_operator == MMINUS_OPER ) ||
             ( last_operator == MMULT_OPER ) ) {
            strcpy( response_message,
                    "Cannot mix scalar variables and matrix operators" );
            return( INVALID_EXPRESSION );
        }

    /*
     *  A valid expression was found, return an indication.
     */

    return( VALID_EXPRESSION );
}
else {

    /*
     *  If the operation is an assignment, verify that the LHS and the RHS are both
     *  of type matrix (i.e. do not allow the assignment of a matrix to a scalar).
     */

    if ( ( number_operators == 0 ) &&
         ( ( lhs_current_attributes & MATRIX ) == 0 ) &&
         ( rhs_current_attributes & MATRIX ) ) {
            strcpy( response_message,
                    "Can not assign a matrix to a scalar variable" );
            return( INVALID_EXPRESSION );
    }

    /*
     *  Mask off the matrix attributes and check the basic data type (if the last
     *  operator was IDENT then no data type will be on the RHS so the check is
     *  ignored).
     */

    lhs_current_attributes &= ~MATRIX;
    rhs_current_attributes &= ~MATRIX;
    if ( ( rhs_current_attributes != lhs_current_attributes ) &&
         ( last_operator != IDENT_OPER ) ) {
            strcpy( response_message,
                    "Matrix operators must match EXACT data type" );
            return( INVALID_EXPRESSION );
    }

    /*
     *  Make sure that only a single operator was specified with the matrix operator
s.
     */

    if ( number_operators > 1 ) {
        strcpy( response_message,
                "Only a single matrix operator is currently supported" );
        return( INVALID_EXPRESSION );
    }

    /*
     *  At this point, we know that we have at most one operator for a matrix
     *  operation. The current implementation of the GCB supports only matrix
     *  assign (i.e. assign a single value to an entire matrix) for matrices with
     *  greater than two dimensions.  Based on this constraint, perform the
     *  following checks.
     */

    if ( ( result.md_num_dimensions > 2 ) ||
         ( operand_1.md_num_dimensions > 2 ) ||
         ( operand_2.md_num_dimensions > 2 ) ) {

        /*
         *  No operators are currently supported.  Generate an error if appropriate.
         */

        if ( number_operators != 0 ) {
            strcpy( response_message,
            "Matrix operators are not current supported for 3D and 4D matrices" );
            return( INVALID_EXPRESSION );
        }

        /*
         *  Determine if the RHS is a simple scalar.
         *
         *  If the operation is a scalar (determined if the attributes of the RHS
         *  first operand are zero or if the attributes of the first operand is a
         *  matrix element), then we simply return (data types have already been
         *  checked).
         */

        if ( ( operand_1.md_attributes == 0 ) ||
             ( ( operand_1.md_attributes & MATRIX ) == 0 ) )
            return( VALID_EXPRESSION );

        /*
         *  At this point, determine if the RHS matrix is the same size as the LHS
         *  matrix.
         */


        if ( ( result.md_num_dimensions != operand_1.md_num_dimensions ) ||
             ( result.md_subs[ SUB1 ] != operand_1.md_subs[ SUB1 ] ) ||
             ( result.md_subs[ SUB2 ] != operand_1.md_subs[ SUB2 ] ) ||
             ( result.md_subs[ SUB3 ] != operand_1.md_subs[ SUB3 ] ) ||
             ( ( result.md_num_dimensions == 4 ) &&
               ( result.md_subs[ SUB4 ] != operand_1.md_subs[ SUB4 ] ) ) ) {
            strcpy( response_message,
            "Matrices must be identically declared when assigned to each other" );
            return( INVALID_EXPRESSION );
        }
        else
            return( VALID_EXPRESSION );
    }

    /*
     *  At this point we know that we either have only one or two dimensional arrays.
     *  The necessary checks and balances are performed.
     *
     *  For matrix add, subtract, assign or multiple perform range cheching on the
     *  involved matrices.
```

```
     *
     *  First check for a simple assignment.  We will check for either a matrix
     *  assignment or a simple scalar assignment.
     */

    if ( number_operators == 0 ) {

        /*
         *  If the operation is a scalar (determined if the attributes of the RHS
         *  first operand are zero or if the attributes of the first operand is a
         *  matrix element), then we simply return (data types have already been
         *  checked).
         */

        if ( ( operand_1.md_attributes == 0 ) ||
             ( ( operand_1.md_attributes & MATRIX ) == 0 ) )
            return( VALID_EXPRESSION );

        /*
         *  The size of the result and operand matrix must be indentical.
         */

        if ( ( result.md_subs[ SUB1 ] != operand_1.md_subs[ SUB1 ] ) ||
             ( result.md_subs[ SUB2 ] != operand_1.md_subs[ SUB1 ] ) ) {
            strcpy( response_message,
                "Matrices assigned to each other must be identically declared" );
            return( INVALID_EXPRESSION );
        }
        else
            return( VALID_EXPRESSION );
    }

    /*
     *  If the operation is either and add or subtract make sure that all sizes are
     *  equal.
     */

    if ( ( last_operator == MADD_OPER ) || ( last_operator == MMINUS_OPER ) ) {

        /*
         *  The size of the result and operand matrix must be indentical.
         */

        if ( ( result.md_subs[ SUB1 ] != operand_1.md_subs[ SUB1 ] ) ||
             ( result.md_subs[ SUB2 ] != operand_1.md_subs[ SUB2 ] ) ||
             ( operand_1.md_subs[ SUB1 ] != operand_2.md_subs[ SUB1 ] ) ||
             ( operand_1.md_subs[ SUB2 ] != operand_2.md_subs[ SUB2 ] ) ) {
            strcpy( response_message,
            "Matrix ADDs or SUBTRACTs must have variables identically declared" );
            return( INVALID_EXPRESSION );
        }
        else
            return( VALID_EXPRESSION );
    }

    /*
     *  If the operation is either and add or subtract make sure that all sizes are
     *  equal.
     */

    if ( last_operator == MMULT_OPER ) {

        /*
         *  The size of the result must be the same as the number of rows of the
         *  first operand and the number of columns of the second operand.
         *  Additionally, the number of columns of the first operand must equal the
         *  number of rows in the second operand.
         */

        if ( ( result.md_subs[ SUB1 ] != operand_1.md_subs[ SUB1 ] ) ||
             ( result.md_subs[ SUB2 ] != operand_2.md_subs[ SUB2 ] ) ||
             ( operand_1.md_subs[ SUB2 ] != operand_2.md_subs[ SUB1 ] ) ) {
            strcpy( response_message,
                "Matrix multiplies must meet common linear algebra guidelines" );
            return( INVALID_EXPRESSION );
        }
        else
            return( VALID_EXPRESSION );
    }

    /*
     *  Check to see if the last operator was a transpose operator.
     */

    if ( last_operator == TRANS_OPER ) {

        /*
         *  Transpose is defined when the number or columns in the source matrix is
         *  equal to the number of columns in the destination and that the number
         *  of rows in the source matrix is equal to the number of columns in the
         *  destination matrix.
         */

        if ( ( result.md_subs[ SUB1 ] != operand_1.md_subs[ SUB2 ] ) ||
             ( result.md_subs[ SUB2 ] != operand_1.md_subs[ SUB1 ] ) ) {
            strcpy( response_message,
                    "the number of rows and columns do not match correctly" );
            return( INVALID_EXPRESSION );
        }
        else
            return( VALID_EXPRESSION );
    }

    /*
     *  Check to see if the last operator was an inverse operator.
     */

    if ( last_operator == INVER_OPER ) {

        /*
         *  Inverse is only defined if the destination matrix has the same number
         *  of rows and the source matrix columns and the destination matrix has
         *  the same number of columns as the source matrix rows.
         */

        if ( ( result.md_subs[ SUB1 ] != operand_1.md_subs[ SUB2 ] ) ||
             ( result.md_subs[ SUB2 ] != operand_1.md_subs[ SUB1 ] ) ) {
            strcpy( response_message,
                "inverse is defined only with source rows = cols and vice versa" );
            return( INVALID_EXPRESSION );
        }
        else
            return( VALID_EXPRESSION );
    }

    /*
     *  Check to see if the last operator was a identity operator.
     */
```

```
    if ( last_operator == IDENT_OPER ) {

        /*
         *  The identity operator is only defined for square matrices.
         */

        if ( result.md_subs[ SUB1 ] != result.md_subs[ SUB2 ] ) {
            strcpy( response_message,
                    "the matrix must be square for the IDENT operator" );
            return( INVALID_EXPRESSION );
        }
        else
            return( VALID_EXPRESSION );
    }

    /*
     *  Check to see if the last operator was a cross operator.
     */

    if ( last_operator == CROSS_OPER ) {

        /*
         *  Cross product is only valid on two vectors with three rows.
         */

        if ( (result.md_subs[ SUB1 ] != 3) || (operand_1.md_subs[ SUB1 ] != 3) ||
             (operand_2.md_subs[ SUB1 ] != 3) || (result.md_subs[ SUB2 ] != 1) ||
             (operand_1.md_subs[ SUB2 ] != 1) || (operand_2.md_subs[ SUB2 ] != 1) )
        {
            strcpy( response_message,
                    "cross product is only defined fro three row vectors" );
            return( INVALID_EXPRESSION );
        }
        else
            return( VALID_EXPRESSION );
    }

    /*
     *  Check to see if the last operator was a dot product operator.
     */

    if ( last_operator == DOT_OPER ) {

        /*
         *  Dot product is only valid for two identically sized vectors.
         */

        if ( (operand_1.md_subs[ SUB1 ] != operand_2.md_subs[ SUB1 ]) ||
             (operand_1.md_subs[ SUB2 ] != 1) || (operand_2.md_subs[ SUB2 ] != 1) )
        {
            strcpy( response_message,
                    "dot product is defined for identically sized vectors" );
            return( INVALID_EXPRESSION );
        }
        else
            /*
             *  The LHS cannot be a matrix.
             */

            if ( result.md_attributes & MATRIX ) {
                strcpy( response_message, "LHS of dot product must be a scalar" );
                return( INVALID_EXPRESSION );
            }
            else
                return( VALID_EXPRESSION );
        }

        /*
         *  At this point we can not identify the operation to verify it so an error is
         *  generated.
         */

        strcpy( response_message, "Expression not currently supported by the GCB" );
        return( INVALID_EXPRESSION );
    }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  return_matrix_data()
*
*
* MODULE FUNCTION:
*
*  Function return_matrix_data will invoke verify_expression to fill the various
*  matrix data attribute structures.  When the function is invoked no errors should
*  be generated as the GCB will not allow code generation in the event of syntax
*  errors.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

int return_matrix_data( expression, operator, result_mat, operand_mat1, operand_mat2 )

    char *expression;
    int *operator;
    struct matrix_data *result_mat, *operand_mat1, *operand_mat2;

{
    char parse_error[ MAX_PARSE_MESSAGE ];

    /*
     *  Invoke verify_expression to fill the various local data structures.
     *  The parse_error variable should be NULL, if not then a syntax error occurred
     *  which MUST be resolved before allowing compilation to occur; return the
     *  appropriate diagnostic.
     */

    if ( verify_expression_type( expression, parse_error ) == INVALID_EXPRESSION )
        return( ERR );

    /*
     *  Copy the appropriate variables to be returned to the build routine.
     */

    memcpy( result_mat, result, sizeof( struct matrix_data ) );
    memcpy( operand_mat1, operand_1, sizeof( struct matrix_data ) );
    memcpy( operand_mat2, operand_2, sizeof( struct matrix_data ) );
    *operator = last_operator;

    /*
     *  Return the type of operation (either scalar or matrix).
     */

    return( parse_status );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  matrix_in_expr()
*
*
* MODULE FUNCTION:
*
*  Function matrix_in_expr will determine if a matrix variable (with no subscripts
*  exists within an expression).  If it does then -1 (ERR) will be returned, otherwise
*  a zero (SUCCESS) will be returned.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

int matrix_in_expr( expression )

    char *expression;

{
    /*
     *  Initialize the matrix_found variable to zero.  Each time a matrix identifier is
     *  found the variable will be incremented by two, each time a left bracket is found,
     *  the variable will be decremented by one.  If at the end of parsing the variable
     *  equals zero, we know that no matrix identified (without an element access) exists
     *  in the expression.
     */

    matrix_found = 0;

    /*
     *  Invoke parse_expression to compute the value of matrix_found.
     */

    parse_expression( expression );

    /*
     *  If the value of matrix_found is zero, no matrix identified exists (with no
     *  element access specified); therefore return success, otherwise an error is
     *  returned.
     */

    if ( matrix_found == 0 )
        return( SUCCESS );
    else
        return( ERR );
}
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    displayer.c
*
*
* FILE FUNCTION:
*
*   This file contains the routines which build and control the "Displayer".  The
*   "Displayer" is a larger NON-modal popup which is used to display results to
*   the user during Element/Comp Installation and during Comp Call Flow.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /home/project/3531/Docu/GCB.spec.doc
*
*
* FILE MODULES:
*
*   build_displayer_popup() - build the Displayer popup
*   cbr_displayer()         - callback for Displayer CANCEL button
*   close_displayer()       - takes down the Displayer popup
*   displayer()             - insert a new string into the Displayer text widget
*   show_displayer()        - clear the Displayer text widget and pop it on the screen
*
*****************************************<----->*****************************************/

#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/Shell.h>
#include <Xm/Xm.h>
#include "gcb.h"
#include "cbr.h"
#include "widgets.h"
#include "constants.h"
#include "displayer.h"
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  build_displayer_popup()
*
* MODULE FUNCTION:
*
*   This routine builds the Displayer popup.  This routines builds the popup and
*   all children widgets.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                          \-  Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

Widget  build_displayer_popup( parent )

    Widget parent;

{
    Arg     args[1];

    /*
     *  Build the Displayer popup.  Set the pointer control style to MODELESS
     *  so the user doesn't have to take the Displayer window down between
     *  operations -> source file builds.
     */

    dlg_displayer = cr_popup( NULLS, parent, "Displayer" );
    XtSetArg( args[0], XmNdialogStyle, XmDIALOG_MODELESS );
    XtSetValues( dlg_displayer, args, 1 );

    FormW = cr_form( NULLS, dlg_displayer, NULL, NULL );
    set_attribs( FORM, FormW, 725, 700, XmRESIZE_NONE );

    scr_displayer = cr_scr_text( NULLS, FormW, 45, False, 675, 10, 10 );

    CancelW = cr_command( NULLS, FormW, "Close", cbr_displayer, 0 );
    HelpW   = cr_command( NULLS, FormW, "Help",  cbr_help, DISPLAYER_HELP );

    set_position( CancelW, 95, IGNORE, 10, IGNORE );
    set_position( HelpW,   95, IGNORE, 75, IGNORE );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_displayer()
*
* MODULE FUNCTION:
*
*   This routine processes the Displayer CANCEL button.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_displayer( w, closure, call_data )

    Widget  w;
    int     closure;
    caddr_t call_data;

{
    close_displayer();
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  close_displayer()
*
* MODULE FUNCTION:
*
*   This routine takes down the Displayer.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  close_displayer()
{
    XtUnmanageChild( dlg_displayer );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  displayer()
*
* MODULE FUNCTION:
*
*   This routine builds a formatted text string and inserts it into the Displayer for
*   viewing.  The specified string is appended to any data already in the text
*   widget.  This routine may also append the string to the Displayer disk file so the
*   user has a copy of the Displayer output on disk.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  displayer( format_str, arg1, arg2 )

    char *format_str,
         *arg1,
          arg2;

{
    FILE    *fp;
    char    string[256];

    sprintf( string, format_str, arg1, arg2 );
    XmTextInsert( scr_displayer, XmTextGetLastPosition(scr_displayer), string );
    XFlush( display );

    /*
     *  If the write-disk-file flag is on, write the same formatted string to
     *  the disk file.
     */

    if ( WriteFile )
        {
        if ((fp = fopen(DisplayFile,"a")) == NULL)
            return;
        fprintf( fp, format_str, arg1, arg2 );
        fclose( fp );
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  show_displayer()
*
* MODULE FUNCTION:
*
*   This routine clears the Displayer's text widget, and then manages the popup to
*   make sure it is popped up on the screen.  The routine can be called to clear
*   the text widget even while the popup is visible (mapped) on the screen.
*
*   This routine may also attempt to open the Displayer disk file.  This will
*   erase the old copy of the file and start a new one every time show_displayer()
*   is called.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  show_displayer( write_file )

    int write_file;

{
    Arg     args[1];
    FILE    *fp;

    XtSetArg     ( args[0], XmNvalue, NULLS );
    XtSetValues  ( scr_displayer, args, 1 );
    XtManageChild( dlg_displayer );

    WriteFile = write_file;
    if ( WriteFile )
        {
        if (! (fp = fopen(DisplayFile,"w")) )
            elog(1,"show_displayer: Couldn't open %s for writing",DisplayFile);
        fclose( fp );
        }
}
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    displayer.h
*
* FILE FUNCTION:
*
*   This header file defines the function prototypes and global variables for the
*   routines in displayer.c
*
*
* FILE MODULES:
*
*   N/A
*
*****************************************<----->*****************************************/


int WriteFile;          /* signals whether displayer() output should be written to
                           the Displayer disk file                                    */


/*
 *  Function prototypes.
 */


Widget          build_displayer_popup();

XtCallbackProc  cbr_displayer();

void            close_displayer(),
                displayer(),
                show_displayer();
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  edit_text()
*
* MODULE FUNCTION:
*
*   This routine handles events when the user pushes the right mouse button
*   while in a symbol (right button edit).  This allows the user to either
*   edit the expression or select a new Element as the target for a SET
*   or CALL symbol.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>

#include "gcb.h"
#include "widgets.h"
#include "element_file.h"
#include "symbol.h"
#include "gcb_parse.h"


void  edit_text( symbol )

    Widget symbol;

{
    Arg         args[1];
    Symbol      *map_entry = (Symbol *) get_sym_map_entry(symbol);
    int         symbol_type;
    XmString    eq_string;

    /*
     *  Determine the type of symbol the user pushed the right
     *  button in.
     */

    XtSetArg( args[0], XmNuserData, &symbol_type );
    XtGetValues( symbol, args, 1 );

    /*
     *  There are no editable attributes in BEGIN or END symbols so
     *  check to see if the user is in one of these symbols.
     */

    if ((symbol_type == BEGIN) || (symbol_type == END))
        {
        user_ack("There are no editable attributes for this symbol");
        return;
        }

    set_current_sym( symbol );

    Mode = EditSymbolContents;

    /*
     *  Determine which type of symbol the user is in.  Show the proper
     *  popup for this type of symbol.
     *
     *  Put up text and expression entry panels; fill the text entry
     *  panels with existing text stored in the Symbol_Map entry
     *  for the parameter symbol.
     */

    switch( symbol_type )
        {
        case IF:
        case SET:

            /*
             *  Set "equals" button to '=' or ':=' according to whether we are
             *  in a SET or an IF.
             */

            XtSetArg(args[0], XmNlabelString, &eq_string);
            XtGetValues(eq_btn, args, 1);
            XmStringFree(eq_string);

            if ( symbol_type == IF )
                eq_string = XmStringCreate( "=",   XmSTRING_DEFAULT_CHARSET );
            else
                eq_string = XmStringCreate( ":=",   XmSTRING_DEFAULT_CHARSET );

            XtSetArg( args[0], XmNlabelString, eq_string );
            XtSetValues( eq_btn, args, 1);
            XmStringFree( eq_string );

            XmTextSetString( scr_logic,   map_entry->Sym.IfSym.logical_expr );
            XmTextSetString( scr_expr,    map_entry->Sym.IfSym.comp_expr );
            XmTextSetString( scr_comment, map_entry->Sym.IfSym.comment );

            /*
             *  set global variable SetSym so parser will know what kind of
             *  expr to expect.
             */

            if ( symbol_type == SET )
                SetSym = 1;
            else SetSym = 0;

            /*
             *  If the symbol expression is not blank, set the insertion
             *  point to the end of the expression so any new text will be
             *  added to the end of the expression.
             */

            if ( map_entry->Sym.IfSym.comp_expr )
                {
                XmTextSetInsertionPosition( scr_expr,
                    strlen(map_entry->Sym.IfSym.comp_expr) );
                /*
                 *  Parse the expression and set the token expression building
                 *  buttons based on the expression in the symbol.
                 */

                do_parse();
                }

            /*
             *  The expression is blank, initialize the expression token buttons
```

```
         *  based on the symbol type -> SET or IF.
         */

        else
            {
            if ( symbol_type == IF )
                init_inputs( -1 );
            else
                init_inputs( -2 );
            }


        /*
         *  Set the active/inactive state of the expression token buttons.
         */

        invalidate_buttons();

        /*
         *  Take down the symbol palette menu and show the expression token
         *  buttons palette menu.
         */

        XtUnmapWidget( frame_palette );
        XtMapWidget( frame_math_menu );

        /*
         *  Show the IF/SET popup.
         */

        XtManageChild( dlg_logic );
        break;

    case PRINT:
    case TEXT:

        /*
         *  Set the popup header string based on whether the user
         *  is editing label text or a PRINT symbol.
         */

        if ( symbol_type == PRINT )
            XtSetArg( args[0], XmNlabelString, print_header );
        else
            XtSetArg( args[0], XmNlabelString, text_header );
        XtSetValues( lbl_text_header, args, 1);

        /*
         *  Set the text in the popup and then show the popup.
         */

        XmTextSetString( scr_text, map_entry->text );
        XtManageChild( dlg_text );
        break;

    case PAUSE:

        XmTextSetString( txt_pause_value, map_entry->text );
        XtManageChild( dlg_pause );
        break;

    case GOTO:

        /*
         *  Load the popup selection list with Element names so the user
         *  can pick a new one from the list.
         */

        if (! load_element_list(ELEMENT,list_elem) )
            {
            XmTextSetString( txt_call_ne, map_entry->text );
            XtManageChild( dlg_call );
            }
        break;

    case START:
    case STOP:

        if (! load_element_list(COMP,list_comp) )
            {
            XmTextSetString( txt_start, map_entry->text );
            XtManageChild( dlg_start );
            }
        break;

    default:
        elog(1,"Invalid symbol key during right button edit: %d", symbol_type);
        return;
    }
}
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    element_file.c
*
*
* FILE FUNCTION:
*
*   Contains the routines which read, write and open Graphical Comp Element Files.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /home/project/3531/Docu/GCB.spec.doc
*
*
* FILE MODULES:
*
*   build_copy_elem_popup() - build the Copy   Element popup
*   build_cre_elem_popup()  - build the Create Element popup
*   build_del_elem_popup()  - build the Delete Element popup
*   build_sel_elem_popup()  - build the Select Element popup
*   build_sel_root_elem()   - build the Select Root Element popup
*   cbr_create_elem_copy()  - process Element copy, create a new Element file
*   cbr_element_type()      - process the Element type toggle buttons -> comp/library
*   cbr_elem_popup()        - determine which Element popup to display to the user
*   cbr_el_delete()         - process Delete Element, purge Element from disk and comp
*   cbr_el_del_sel()        - process the selection widget selection, load text widget
*   cbr_el_selected()       - user has selected a new Element file, read it in
*   cbr_new_element()       - creates a new Element file on disk
*   cbr_root_elem()         - process the Select Root Element popup
*   cbr_root_selected()     - process the selection widget selection, load text widget
*   cbr_sav_element()       - user has selected Save Element
*   get_element_calls()     - extracts the calls to other Elements from an Element
*   get_element_selfcalls() - extracts the number of times an Element calls itself
*   implode()               - process the Implode button press
*   init_element_vars()     - initialize the Element level global variables
*   load_element_list()     - load a selection list widget with Element names
*   locate_line()           - locate a line     structure during Element file reading
*   locate_list()           - locate a LineList structure during Element file reading
*   locate_segment()        - locate a segment  structure during Element file reading
*   locate_symbol()         - locate a symbol   structure during Element file reading
*   read_element_file()     - read an Element file from disk, restore pointers and vars
*   read_element_str()      - read a string from an Element file, read a count of chars
*   record_line()           - build a line struct during the saving of an Element file
*   record_list()           - build a list struct during the saving of an Element file
*   record_segment()        - build a seg  struct during the saving of an Element file
*   reinit_element_vars()   - reinitialize Element level global vars
*   save_curr_element()     - process the user's Save Element button press
*   save_element_file()     - write an Element file to disk, save all pointers and vars
*   save_element_str()      - save a string to an Element file, save a count of chars
*
*****************************************<----->*****************************************/

#include <stdio.h>
#include <stdlib.h>
#include <sys/file.h>
#include <sys/time.h>
#include <dirent.h>

#include <X11/Intrinsic.h>
#include <Xm/Xm.h>

#include "gcb.h"
#include "cbr.h"
#include "widgets.h"
#include "element_file.h"
#include "lines.h"
#include "pixmaps.h"
#include "constants.h"
#include "fonts.h"
#include "symbol.h"
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  build_copy_elem_popup()
*
* MODULE FUNCTION:
*
*   This routine builds the copy element popup.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

Widget  build_copy_elem_popup( parent )

    Widget parent;

{
    Arg     args[2];

    dlg_copy_elem = cr_popup( NULLS, parent, "Copy Element" );

    FormW = cr_form( NULLS, dlg_copy_elem, NULL, NULL );
    set_attribs( FORM, FormW, 450, 200, XmRESIZE_NONE );

    rb_copy_elem     = cr_radio_box( NULLS, FormW, XmHORIZONTAL );
    tgl_copy_elem_ce = cr_toggle( NULLS, rb_copy_elem, "Comp Element",    NULL,0,0 );
    tgl_copy_elem_le = cr_toggle( NULLS, rb_copy_elem, "Library Element", NULL,0,0 );

    cr_label( NULLS, FormW, "New Element Name:",0,35,IGNORE,9,IGNORE );
    cr_label( NULLS, FormW, "Element Type:",    0,12,IGNORE,9,IGNORE );

    txt_copy_elem_ne = cr_text  ( NULLS, FormW, NULL, NULL, NULLS, FALSE, 1, 37 );

    cr_separator( NULLS, FormW, 75, IGNORE, 1, 99 );

    DoneW   = cr_command( NULLS, FormW, "Create",cbr_create_elem_copy, DONE );
    CancelW = cr_command( NULLS, FormW, "Cancel",cbr_create_elem_copy, CANCEL );
    HelpW   = cr_command( NULLS, FormW, "Help",  cbr_help,             COPY_ELEM );

    set_position( rb_copy_elem,      9, IGNORE, 40, IGNORE );
    set_position( txt_copy_elem_ne, 35, IGNORE, 40, IGNORE );
    set_position( CancelW,          84, IGNORE,  5, IGNORE );
    set_position( DoneW,            84, IGNORE, 40, IGNORE );
    set_position( HelpW,            84, IGNORE, 75, IGNORE );

    XtSetArg( args[0], XmNshowAsDefault, 2 );
    XtSetArg( args[1], XmNdefaultButtonShadowThickness, 0 );
    XtSetValues( CancelW, args, 2 );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  build_cre_elem_popup()
*
* MODULE FUNCTION:
*
*   This routine builds the create element popup.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

Widget  build_cre_elem_popup( parent )

    Widget parent;

{
    dlg_cre_elem = cr_popup( NULLS, parent, "Create Element" );

    FormW = cr_form( NULLS, dlg_cre_elem, NULL, NULL );
    set_attribs( FORM, FormW, 450, 400, XmRESIZE_NONE );

    rb_cre_elem         = cr_radio_box( NULLS, FormW, XmHORIZONTAL );

    tgl_cre_elem_ce = cr_toggle( NULLS, rb_cre_elem, "Comp Element",    NULL,0,0 );
    tgl_cre_elem_le = cr_toggle( NULLS, rb_cre_elem, "Library Element", NULL,0,0 );
    txt_cre_elem_ne = cr_text  ( NULLS, FormW, NULL, NULL, NULLS, FALSE, 1, 37);

    cr_label( NULLS, FormW, "Element Type:", 0,  5, IGNORE, 9, IGNORE );
    cr_label( NULLS, FormW, "New Element:",  0, 17, IGNORE, 9, IGNORE );
    cr_label( NULLS, FormW, "Purpose:",      0, 30, IGNORE, 9, IGNORE );

    scr_cre_elem = cr_scr_text(NULLS, FormW, 10, True, 300, NULL, NULL);

    cr_separator( NULLS, FormW, 80, IGNORE, 1, 99 );

    DoneW   = cr_command( NULLS, FormW, "Create", cbr_new_element, DONE );
    CancelW = cr_command( NULLS, FormW, "Cancel", cbr_new_element, CANCEL );
    HelpW   = cr_command( NULLS, FormW, "Help",   cbr_help,        CREATE_ELEM);

    set_position( XtParent(scr_cre_elem),   29, IGNORE, 30, IGNORE );
    set_position( txt_cre_elem_ne,          17, IGNORE, 30, IGNORE );
    set_position( rb_cre_elem,               4, IGNORE, 33, IGNORE );
    set_position( CancelW,                  88, IGNORE,  5, IGNORE );
    set_position( DoneW,                    88, IGNORE, 40, IGNORE );
    set_position( HelpW,                    88, IGNORE, 75, IGNORE );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  build_del_elem_popup()
*
* MODULE FUNCTION:
*
*   This routine builds the delete element popup.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

Widget  build_del_elem_popup( parent )

    Widget parent;

{
    dlg_del_elem = cr_popup( NULLS, parent, "Delete Element" );

    FormW = cr_form( NULLS, dlg_del_elem, NULL, NULL );
    set_attribs( FORM, FormW, 425, 500, XmRESIZE_NONE );

    cr_label( NULLS, FormW, "Delete Element:",0,20,23,9,IGNORE );
    cr_label( NULLS, FormW, "Element List:",  0,30,33,9,IGNORE );
    cr_label( NULLS, FormW, "Element Type:",  0,10,13,9,IGNORE );

    txt_del_el    = cr_text( NULLS, FormW, NULL, NULL, NULLS, FALSE, 1, 34);
    list_del_elem = cr_list( NULLS, FormW, 14, 235 );

    rb_del_elem = cr_radio_box( NULLS, FormW, XmHORIZONTAL );

    tgl_del_elem_ce = cr_toggle( NULLS, rb_del_elem,"Comp Element", NULL,0,0 );
    tgl_del_elem_le = cr_toggle( NULLS, rb_del_elem, "Library Element", NULL, 6, 6);

    set_user_data( tgl_del_elem_ce, ELEMENT );
    set_user_data( tgl_del_elem_le, LIB );

    cr_separator( NULLS, FormW, 86, IGNORE, 1, 99 );

    DoneW   = cr_command( NULLS, FormW, "Delete", cbr_el_delete, DONE );
    CancelW = cr_command( NULLS, FormW, "Cancel", cbr_el_delete, CANCEL );
    HelpW   = cr_command( NULLS, FormW, "Help",   cbr_help,      DELETE_ELEM );

    XtAddCallback( list_del_elem,   XmNbrowseSelectionCallback, cbr_el_del_sel, NULL);
    XtAddCallback( tgl_del_elem_le, XmNarmCallback, cbr_element_type, list_del_elem);
    XtAddCallback( tgl_del_elem_ce, XmNarmCallback, cbr_element_type, list_del_elem);

    set_position( txt_del_el,              19, IGNORE, 40, IGNORE );
    set_position( XtParent(list_del_elem), 29, IGNORE, 40, IGNORE );
    set_position( rb_del_elem,              9, IGNORE, 40, IGNORE );
    set_position( CancelW,                 92, IGNORE,  5, IGNORE );
    set_position( DoneW,                   92, IGNORE, 40, IGNORE );
    set_position( HelpW,                   92, IGNORE, 75, IGNORE );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  build_sel_elem_popup()
*
* MODULE FUNCTION:
*
*   This routine builds the select element popup.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

Widget  build_sel_elem_popup( parent )

    Widget parent;

{
    dlg_sel_elem = cr_popup( NULLS, parent, "Select Element" );

    FormW = cr_form( NULLS, dlg_sel_elem, NULL, NULL );
    set_attribs( FORM, FormW, 425, 400, XmRESIZE_NONE );

    cr_label( NULLS,FormW,"Element Type: ",0, 5,IGNORE,9,IGNORE );
    cr_label( NULLS,FormW,"Element List:", 0,17,IGNORE,9,IGNORE );

    rb_sel_elem = cr_radio_box( NULLS, FormW, XmHORIZONTAL );

    tgl_sel_elem_ce = cr_toggle( NULLS, rb_sel_elem, "Comp Element",    NULL, 3,3 );
    tgl_sel_elem_le = cr_toggle( NULLS, rb_sel_elem, "Library Element", NULL, 2,2 );

    set_user_data( tgl_sel_elem_ce, ELEMENT );
    set_user_data( tgl_sel_elem_le, LIB );

    list_sel_elem = cr_list( NULLS,FormW, 14, 235 );

    cr_separator( NULLS, FormW, 88, IGNORE, 1, 99 );

    CancelW = cr_command( NULLS, FormW, "Cancel", cbr_el_selected, CANCEL );
    HelpW   = cr_command( NULLS, FormW, "Help",   cbr_help,        SELECT_ELEM );

    set_position( XtParent(list_sel_elem),  17, IGNORE, 40, IGNORE );
    set_position( rb_sel_elem,               4, IGNORE, 40, IGNORE );
    set_position( CancelW,                  93, IGNORE, 10, IGNORE );
    set_position( HelpW,                    93, IGNORE, 70, IGNORE );

    XtAddCallback(list_sel_elem,XmNbrowseSelectionCallback,cbr_el_selected,list_sel_elem)
;
    XtAddCallback( tgl_sel_elem_le,     XmNarmCallback, cbr_element_type, list_sel_elem);
    XtAddCallback( tgl_sel_elem_ce,     XmNarmCallback, cbr_element_type, list_sel_elem);
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  build_sel_root_elem()
*
* MODULE FUNCTION:
*
*   This routine builds the popup which allows the user to select a new root
*   element.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

Widget  build_sel_root_elem( parent )

    Widget parent;

{
    dlg_sel_root = cr_popup( NULLS, parent, "Select Root Element" );

    FormW = cr_form( NULLS, dlg_sel_root, NULL, NULL );
    set_attribs( FORM, FormW, 425, 400, XmRESIZE_NONE );

    cr_label( NULLS, FormW, "Root Element:", 0,  5,IGNORE, 9,IGNORE );
    cr_label( NULLS, FormW, "Element List:", 0, 20,IGNORE, 9,IGNORE );

    txt_sel_root  = cr_text( NULLS, FormW, NULL, NULL, NULLS, FALSE, 1, 37);
    list_sel_root = cr_list( NULLS, FormW, 14, 235 );

    cr_separator( NULLS, FormW, 88, IGNORE, 1, 99 );

    DoneW   = cr_command( NULLS, FormW, "Done",   cbr_root_elem, DONE );
    CancelW = cr_command( NULLS, FormW, "Cancel", cbr_root_elem, CANCEL );
    HelpW   = cr_command( NULLS, FormW, "Help",   cbr_help,      SEL_ROOT_ELEM );

    set_position( XtParent(list_sel_root),  20, IGNORE, 40, IGNORE );
    set_position( txt_sel_root,              5, IGNORE, 40, IGNORE );
    set_position( CancelW,                  92, IGNORE,  3, IGNORE );
    set_position( DoneW,                    92, IGNORE, 40, IGNORE );
    set_position( HelpW,                    92, IGNORE, 75, IGNORE );

    XtAddCallback( list_sel_root,XmNbrowseSelectionCallback,cbr_root_selected,LIST_SEL);
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_create_elem_copy()
*
* MODULE FUNCTION:
*
*   This routine processes the user's button presses in the Copy Element popup.  This
*   routine will create a copy of the current Element file if the user choses DONE.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.01 - 08/01/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_create_elem_copy( w, closure, call_data )

    Widget  w;
    int     closure;
    caddr_t call_data;

{
    FILE    *fp;
    char    copy_ok_msg[100],
            PrevElementFile[MAX_NAME];

    /*
     *  User wants to CANCEL and exit the popup.
     */

    if ( closure == CANCEL )
        {
        XtUnmanageChild( dlg_copy_elem );
        return;
        }

    /*
     *  Make sure we have a valid element, there is no need to copy if we don't
     *  have a valid element.
     */

    if (! ValidElement )
        {
        user_ack("Can't copy, invalid Element file");
        elog(1,"cbr_create_elem_copy: Can't copy, invalid Element file");
        XtUnmanageChild( dlg_copy_elem );
        return;
        }

    if ( SaveNeeded )
        if ( ask("Do you want to save your changes to the current element file?"))
            save_element_file();

    /*
     *  Check to make sure the user entered a new Element file name.
     *
     *  copy popup's element file name to elementfile and gelementfile;
     *  element type to ElementType.
     */
```

```
if (! XmTextGetInsertionPosition(txt_copy_elem_ne) )
    {
    user_ack("Can't copy element, no new element file name specified");
    XtUnmanageChild( dlg_copy_elem );
    return;
    }

busy( dlg_copy_elem, TRUE );

/*
 *  Save the current Element info before getting the new Element name.
 */

strcpy( PrevElementFile, ElementFile );
strcpy( ElementFile, (char *) XmTextGetString(txt_copy_elem_ne) );

/*
 *  Determine which type of Element file to create based on the toggle buttons.
 */

if ( XmToggleButtonGetState(tgl_copy_elem_ce) )
    ElementType = ELEMENT;
else
    ElementType = LIB;

/*
 *  If the user wants to build a library element, we need to
 *  add the library path.
 */

if ( ElementType == LIB )
    {
    strcpy( GElementFile, LibPath );
    strcat( GElementFile, "/" );
    strcat( GElementFile, ElementFile );
    strcat( GElementFile, EL_EXT );
    }
else
    {
    strcpy( GElementFile, ElementFile );
    strcat( GElementFile, EL_EXT );
    }

/*
 *  If the file already exists, let the user know.  Clear the Work Area because
 *  we have changed all the Element paths and variables.
 */

if (! access(GElementFile,R_OK) )
    {
    user_ack("Couldn't open the new element file, file already exists");
    reinit_element_vars();
    XtUnmanageChild( dlg_copy_elem );
    return;
    }

/*
 *  Try to open new file, if this fails, clear the Work Area because we have
 *  changed all the Element paths and variables.
 */

if (! (fp = fopen(GElementFile,"w")) )
    {
    user_ack("Couldn't open the new element file for writing");
    elog(1,"cbr_create_elem_copy: Couldn't open the new element file for writing");
    reinit_element_vars();
    XtUnmanageChild( dlg_copy_elem );
    return;
    }
else
    fclose( fp );

/*
 *  save the current file
 */

save_element_file();

/*
 *  If we have a valid Comp file, then update the symbol table by adding the
 *  new Element file to the symbol table.
 */

if ( ValidComp )
    {
    if ( copy_element_to_symbol_table(PrevElementFile) )
        user_ack("Error: could not add new Element to Comp symbol table");

    /*
     *  If we have a Comp file, then add this new element to the Comp file
     *  element list.
     */

    if ( ElementType == LIB )
        update_comp_file( GCompFile, NULL, NULL, ElementFile, "L" );
    else
        update_comp_file( GCompFile, NULL, NULL, ElementFile, "E" );

    /*DEBUG*/
    elog(3,"cbr_create_elem_copy: updated comp");

    strcpy( copy_ok_msg, "Copy made; you are now editing ");
    strcat( copy_ok_msg, ElementFile );
    user_ack( copy_ok_msg );
    }
else
    {
    user_ack("ValidComp not set during cbr_create_elem_copy");
    elog(1,"ValidComp not set during cbr_create_elem_copy");
    }

XtUnmanageChild( dlg_copy_elem );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_element_type()
*
* MODULE FUNCTION:
*
*   This routine responds to the Element Type toggle buttons.  This popup will load
*   the Element selection list with the proper type of element names: either comp
*   elements or library elements based on the user's selection of the toggle buttons.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_element_type( w, client_data, call_data )

    Widget  w;
    caddr_t client_data,
            call_data;

{
    Widget  list;
    Arg     args[1];
    char    tmpStr[MAX_NAME];



    list = (Widget) client_data;

    /*
     *  The user has clicked one of the element type selection buttons,
     *  set the global var to the newly selected type and then reload the
     *  element selection list based on the type.
     */

    XtSetArg( args[0], XmNuserData, &ElementListType );
    XtGetValues( w, args, 1 );

    load_element_list( ElementListType, list );

    /*
     *  Clear the text widget in "delete element".  This will be executed
     *  during "non-delete-element" operations, but it won't hurt.
     */

    XmTextSetString( txt_del_el, NULLS );

    /*
     *  Set the text widget in "create call symbol".  This will be executed
     *  during "non-create_call-element" operations, but it won't hurt.
     */

    sscanf( selList[0], "%s", tmpStr );
    XmTextSetString( txt_call_ne, tmpStr );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_elem_popup()
*
* MODULE FUNCTION:
*
*   This routine pops up the Element popups.  This routine will display one of the
*   following popups:
*                       Create Element
*                       Copy   Element
*                       Delete Element
*                       Select Element
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_elem_popup( w, closure, call_data )

    Widget  w;
    int     closure;
    caddr_t call_data;

{
    if (Mode != EditSymbol)
        return;

    /*
     *  Start all element file operations with the type set to: ELEMENT.
     */

    ElementListType = ELEMENT;

    /*
     *  Determine which element file operation the user wants and then show
     *  the corresponding popup.
     */

    switch ( closure )
        {
        case 0:    arm_tgl( tgl_cre_elem_ce );
                   disarm_tgl( tgl_cre_elem_le );
                   XtManageChild( dlg_cre_elem );
                   busy( dlg_cre_elem, FALSE );
                   break;
        case 1:    arm_tgl( tgl_copy_elem_ce );
                   disarm_tgl( tgl_copy_elem_le );
                   XtManageChild( dlg_copy_elem );
                   busy( dlg_copy_elem, FALSE );
                   break;
        case 2:    arm_tgl( tgl_del_elem_ce );
                   disarm_tgl( tgl_del_elem_le );
                   if (! load_element_list(ELEMENT,list_del_elem) )
                       {
                       XtManageChild( dlg_del_elem );
                       busy( dlg_del_elem, FALSE );
                       }
                   break;
        case 3:    arm_tgl( tgl_sel_elem_ce );
```

```
        disarm_tgl( tgl_sel_elem_le );
        if (! load_element_list(ELEMENT,list_sel_elem) )
            {
            XtManageChild( dlg_sel_elem );
            busy( dlg_sel_elem, FALSE );
            }
        break;
    }
}
```

```
/*****************************************<----->*************************************
*
* MODULE NAME:  cbr_el_delete()
*
* MODULE FUNCTION:
*
*   This routine processes the Delete Element popup and deletes the selected element
*   from the comp file.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*************************************/

XtCallbackProc  cbr_el_delete( w, client_data, call_data )

    Widget                  w;
    int                     client_data;
    XmListCallbackStruct    *call_data;

{
    struct symbol_entry *entry;

    char     el_file[MAX_NAME],     /* element file, no path or extension   */
            gel_file[MAX_PATH];     /* element file, with path and extension*/
    int     self_calls;


    /*
     *  User does not want to delete an element, take down the popup.
     */

    if ( client_data == CANCEL )
        {
        XtUnmanageChild( dlg_del_elem );
        return;
        }

    /*
     *  User didn't specify an element to delete but still selected DELETE.
     */

    if ( strlen(XmTextGetString(txt_del_el)) == 0 )
        {
        user_ack("Please select an element to delete from the list");
        return;
        }

    busy( dlg_del_elem, TRUE );

    /*
     *  Get the selected Element name.
     */

    sscanf( XmTextGetString(txt_del_el), "%s", el_file );

    /*
     *  Build the paths depending on whether the user wants to delete a libary
     *  element or comp element.
     */
```

```
if ( ElementListType == ELEMENT )
    {
    strcpy( gel_file, el_file );
    strcat( gel_file, EL_EXT );
    }
else
    {
    strcpy( gel_file, LibPath );
    strcat( gel_file, "/" );
    strcat( gel_file, el_file );
    strcat( gel_file, EL_EXT );
    }

/*
 *  Get the use count of the element.
 */

if (! (entry = lookup_symbol(NULL,el_file)) )
    elog(1,"cbr_el_delete: Element not found in the symbol table: %s",el_file);
else
    {

    /*
     *  Check the use count of the element to be deleted.
     */

    if ( entry->se_use_count )
        self_calls = get_element_selfcalls( el_file, gel_file );
    else
        self_calls = 0;

    /*
     *  If the use count is greater than the number of selfcalls in the element,
     *  then notify the user that the element has other references and can not
     *  be deleted.
     */

    if ( entry->se_use_count > self_calls )
        {
        user_ack("This element is referenced by another element - can not delete");
        busy( dlg_del_elem, FALSE );
        return;
        }

    /*
     *  Remove the Element and its locals from the symbol table.
     */

    if ( del_elem_from_symbol_table(el_file,gel_file) )
        user_ack("Error: deleting the Element from the Comp symbol table");
    }

 /*
  *  Delete the file.
  */

if ( unlink(gel_file) )
   user_ack("Could not delete element file - file may not have been created");

/*
 *  Remove element from the Comp file.
 */

    del_element_from_comp( el_file, ElementListType );

    /*
     *  Let the user know the element was deleted.
     */

    user_ack("Element file deleted - references in other elements may still exist");
    XmTextSetString( txt_del_el, NULLS );

    /*
     *  If the user deleted the element which was currently in the Work Area,
     *  clear the Work Area and reset the status flags.
     */

    if (! strcmp(gel_file,GElementFile) )
        reinit_element_vars();

    /*
     *  Update the selection list, let the user delete another element.
     */

    if ( XmToggleButtonGetState(tgl_del_elem_le) )
        cbr_element_type( tgl_del_elem_le, list_del_elem, call_data );
    else
        cbr_element_type( tgl_del_elem_ce, list_del_elem, call_data );

    /*
     *  Turn off BUSY here because we don't take down the popup during delete
     *  operations.
     */

    busy( dlg_del_elem, FALSE );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_el_del_sel()
*
* MODULE FUNCTION:
*
*   This routine gets the user's Element name selection from the selection list and
*   inserts it into the Delete Element text string field.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_el_del_sel( w, client_data, call_data )

    Widget                  w;
    caddr_t                 client_data;
    XmListCallbackStruct    *call_data;

{
    char    *string,
            tempName[MAX_NAME];

    XmStringGetLtoR( call_data->item, XmSTRING_DEFAULT_CHARSET, &string );

    sscanf( string, "%s", tempName );
    XmTextSetString( txt_del_el, tempName );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_el_selected()
*
* MODULE FUNCTION:
*
*   This routine responds to the user's selecting an element from the list.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_el_selected( w, closure, call_data )

    Widget                  w;
    int                     closure;
    XmListCallbackStruct    *call_data;

{
    FILE *fp;
    char *string;

    if ( closure == CANCEL )
        {
        XtUnmanageChild( dlg_sel_elem );
        return;
        }

    busy( dlg_sel_elem, TRUE );

    XmStringGetLtoR( call_data->item, XmSTRING_DEFAULT_CHARSET, &string );

    /*
     *  See if the current file has been updated and should be saved.
     */

    if ( SaveNeeded )
        save_curr_element();

    reinit_element_vars();

    /*
     *  Copy element name into ElementFile; set up GElementFile.
     */

    sscanf( string, "%s", ElementFile );

    if ( ElementListType == ELEMENT )
        {
        strcpy( GElementFile, ElementFile );
        strcat( GElementFile, EL_EXT );
        }

    /*
     *  If the user wants to read a library element, prepend the library path
     *  to the element name.
     */

    else
```

```
        {
        strcpy( GElementFile, LibPath );
        strcat( GElementFile, "/" );
        strcat( GElementFile, ElementFile );
        strcat( GElementFile, EL_EXT );
        }

    /*
     *  See if the element file exists, if it doesn't ask the user if they want
     *  to create one.
     */

    if ( access(GElementFile,R_OK) )
        if ( ask("Selected Element file does not exist, create one?") )
            {
               arm_tgl( tgl_cre_elem_ce );
            disarm_tgl( tgl_cre_elem_le );
            XmTextSetString( txt_cre_elem_ne, ElementFile );
            XtManageChild( dlg_cre_elem );
            busy( dlg_cre_elem, FALSE );
            }
        else
            {
            user_ack("Element file not created, please select another");
            ValidElement = False;
            busy( dlg_sel_elem, FALSE );
            return;
            }

    /*
     *  File exists, read it.
     */

    else
        if ( read_element_file() == ERR )
            {
            user_ack("Unable to read Element file - please check error log");
            elog(1,"Unable to read Element file - %s", GElementFile );
            }

    upd_pos_panel( ELEMENT_PURPOSE );
    upd_mode_panel();

    XtUnmanageChild( dlg_sel_elem );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_new_element()
*
* MODULE FUNCTION:
*
*   This routine creates a new Element file.  This routine is called when the user
*   is in the "Create New Element" popup.  This routine processes both the CREATE
*   button and the CANCEL button.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                                  Release 1.01 - 08/01/91
*                                  Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_new_element( w, closure, call_data )

    Widget  w;
    int     closure;
    caddr_t call_data;

{
    FILE    *fp;
    time_t  clock;
    int     attributes = 0;

    if ( closure == DONE )
        {

        if (! strlen(XmTextGetString(txt_cre_elem_ne)) )
            {
            user_ack("Please enter an Element name");
            return;
            }

        /*
         *  See if the current file has been updated and should be saved.
         */

        busy( dlg_cre_elem, TRUE );

        if ( SaveNeeded )
            save_curr_element();

        /*
         *  Clear work area and reinit variables.
         */

        reinit_element_vars();

        /*
         *  Get element file name and get element file type -> comp or library element.
         */

        strcpy( ElementFile, (char *) XmTextGetString(txt_cre_elem_ne) );

        if ( XmToggleButtonGetState(tgl_cre_elem_ce) )
            ElementType = ELEMENT;
        else
```

```
    ElementType = LIB;

/*
 *  If the user wants to build a library element, we need to
 *  add the library path.
 */

if ( ElementType == LIB )
    {
    strcpy( GElementFile, LibPath );
    strcat( GElementFile, "/" );
    strcat( GElementFile, ElementFile );
    strcat( GElementFile, EL_EXT );
    }
else
    {
    strcpy( GElementFile, ElementFile );
    strcat( GElementFile, EL_EXT );
    }

/*
 *  See if an Element file with this name already exists.
 */

if (! access(GElementFile,R_OK) )
    {
    busy( dlg_cre_elem, FALSE );
    user_ack("Couldn't open the new element file, file already exists");
    return;
    }

/*
 *  Try to open the new Element file.
 */

if (! (fp = fopen(GElementFile,"w")) )
    {
    busy( dlg_cre_elem, FALSE );
    user_ack("Couldn't open the new element file for writing");
    elog(1,"Couldn't open the new element file for writing: %s",GElementFile);
    return;
    }
else
    fclose( fp );

/*
 *  If we have a Comp file, then add this new element to the Comp file
 *  element list.
 */

if ( ValidComp )
    if ( ElementType == LIB )
        update_comp_file( GCompFile, NULL, NULL, ElementFile, "L" );
    else
        update_comp_file( GCompFile, NULL, NULL, ElementFile, "E" );

/*
 *  Get the purpose text.
 */

strcpy( ElementPurpose, (char *) XmTextGetString(scr_cre_elem) );

/*
 *  Get the current time and date.
 */

    clock = time( NULL );
    strftime( CreateDate, 11, "%m/%d/%Y", localtime(&clock) );
    strftime( UpdateDate, 11, "%m/%d/%Y", localtime(&clock) );
    strftime( UpdateTime, 9,  "%T",       localtime(&clock) );

    upd_pos_panel( ELEMENT_PURPOSE );
    upd_mode_panel();
    ValidElement = TRUE;

    attributes |= PROCEDURE;
    if (! (struct symbol_entry *) lookup_symbol(NULL,ElementFile) )
        {
        /*DEBUG*/
        elog(3,"cre_elem: adding entry %s to symtable", ElementFile);
        if ( add_symbol_entry(NULL,ElementFile,attributes,0,0,0,0,0) )
            error_handler( ERR_ADD_SYMBOL, "cre_elem" );
        }
    else
        /*DEBUG*/
        elog(3,"cre_elem: entry %s found in symtable, not adding", ElementFile);
    }

/*
 *  User does not want to create a new Element file, take down the popup.
 */

else if ( closure == CANCEL )
    {
    if (! ValidElement )
        {
        GElementFile[0] = NULL;
        ElementFile[0]  = NULL;
        XmTextSetString( scr_cre_elem, "" );
        XmTextSetString( txt_cre_elem_ne, "" );
        }
    }

XtUnmanageChild( dlg_cre_elem );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_root_elem()
*
* MODULE FUNCTION:
*
*   This routine displays the popup which allows the user to select a new root
*   element.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.01 - 08/01/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_root_elem( w, key, call_data )

    Widget  w;
    int     key;
    caddr_t call_data;

{
    if (Mode != EditSymbol)
        return;

    switch ( key )
        {
        case MANAGE :
            load_element_list( ELEMENT, list_sel_root );
            XtManageChild( dlg_sel_root );
            busy( dlg_sel_root, FALSE );
            break;

        case DONE:
            if ( strlen(XmTextGetString(txt_sel_root)) < 1 )
                {
                user_ack("Please select an element to become the root element");
                return;
                }

            /*
             *  Get the new Root Element name and update the global variable.
             */

            strcpy( RootElement, XmTextGetString(txt_sel_root) );

            /*
             *  Update the Comp file which will update the RootElement name.
             */

            update_comp_file( GCompFile, CompPurpose, RootElement, NULL, NULL );

            /*
             *  Take down the popup.
             */

            XtUnmanageChild( dlg_sel_root );
            break;

        case CANCEL :
            XtUnmanageChild( dlg_sel_root );
            break;

        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_root_selected()
*
* MODULE FUNCTION:
*
*   This routine is a callback for the Select Root Element selection list widget.  When
*   the user selects an Element name from the list, this routine is called to insert
*   the string into the Element name single line text field.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_root_selected( w, list, call_data )

    Widget                  w;
    Widget                  list;
    XmListCallbackStruct    *call_data;

{
    char *string,
         tempName[MAX_NAME];


    XmStringGetLtoR( call_data->item, XmSTRING_DEFAULT_CHARSET, &string );

    sscanf( string, "%s", tempName );
    XmTextSetString( txt_sel_root, tempName );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_sav_element()
*
* MODULE FUNCTION:
*
*   This callback routine makes a call to save the current element.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_sav_element( w, list, call_data )

    Widget  w;
    Widget  list;
    caddr_t call_data;

{
    /*
     *  Make sure we really need to save the file.
     */

    if (! SaveNeeded )
        if (! ask("No changes have been made, do you still want to SAVE?") )
            return;

    if (! ValidElement )
        {
        user_ack("A valid file has not been started, no filename, can not save");
        return;
        }

    save_element_file();

    /*DEBUG*/
    elog(3,"cbr save elem: updating comp file %s", GCompFile);

    update_comp_file( GCompFile, CompPurpose, NULL, NULL, NULL );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  get_element_calls()
*
* MODULE FUNCTION:
*
*   This routine extracts the calls to other Elements from the specified Element
*   file.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  get_element_calls( names, types, element_name, element_type )

    char    names[][MAX_NAME],
            types[],
           *element_name,
            element_type;

{
    FILE    *fp;
    char    elementName[MAX_NAME],
            epath[MAX_PATH];
    int     count = 0,
            elementType,
            i,
            rc,
            symbol_type;


    /*
     *  Open the specified element file and locate any calls to other element
     *  files.
     */

    if ( element_type == 'L' )
        {
        strcpy( epath, LibPath );
        strcat( epath, "/" );
        strcat( epath, element_name );
        strcat( epath, EL_EXT );
        }
    else
        {
        strcpy( epath, element_name );
        strcat( epath, EL_EXT );
        }

    if (! (fp = fopen(epath,"r")) )
        {
        elog(3,"Couldn't open an Element file during get_element_calls(): %s", epath );
        return( 0 );
        }

    /*
     *  Locate a graphical symbol which will contain an element file name.
     */

    while ( (rc = fscanf(fp,"%d",&symbol_type)) != EOF )
        {

        /*
         *  If we don't get an integer, dump the rest of the line.  If we
         *  get a LINE SEGMENT indicator, we are finished with the file
         *  because the line info follows the symbol info.
         */

        if ( rc == 0 )
            {
            fscanf( fp, "%*[^\n]" );
            continue;
            }

        if ( symbol_type == SEGMENT_KEY )
            break;

        /*
         *  See if we have a symbol which may contain a reference to an
         *  element file.
         */

        switch ( symbol_type )
            {
            case START:
            case STOP:
            case GOTO:  fscanf( fp,"%*u %*d %*d %*d %*d %*d %*d %*d %*d %*d %*d" );
                        read_element_str( fp, elementName );
                        fscanf( fp, "%*u %*u %d", &elementType );
                        fscanf( fp, "%*[^\n]" );
                        break;
            default :   fscanf( fp, "%*[^\n]" );
                        continue;
            }

        /*
         *  Loop through the list, if the new element name is already in the list,
         *  ignore it, otherwise add the new element name and update the list size.
         */

        i = 0;
        while ( i<count )
            if (! strcmp(names[i],elementName) )
                {
                if ( (types[i] == 'L') && (elementType == LIB) )
                    break;
                else if ( (types[i] == 'E') && (elementType == ELEMENT) )
                        break;
                     else
                        i++;
                }
            else
                i++;

        /*
         *  If we are at the end of the list, this name must be a new
         *  one so add it to the list.
         */

        if ( i == count )
            {
            strcpy( names[count], elementName );
            if ( elementType == LIB )
```

```
                types[count] = 'L';
            else
                types[count] = 'E';
            count++;
            }
        }

    /*
     *  Close the element file.
     */

    fclose( fp );

    return( count );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  get_element_selfcalls()
*
* MODULE FUNCTION:
*
*   This routine opens an Element file and counts the number of times the Element
*   calls itself.
*
*   NOTE: this routine does not do any checking for Comp Element vs. Library
*         Element.  This code will need to be added as the Library Element's
*         impacts on the Symbol Table are decided.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                                  Release 1.01 - 08/01/91
*                                  Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  get_element_selfcalls( el_file, gel_file )

    char     *el_file,      /* element name without path or extension   */
            *gel_file;      /* element name, path and extension         */

{
    FILE    *fp;
    char    elementName[MAX_NAME];
    int     count = 0,
            rc,
            symbol_type;


    if (! (fp = fopen(gel_file,"r")) )
        {
        elog(3,"Couldn't open Element file in get_element_selfcalls(): %s", gel_file );
        return( 0 );
        }

    /*
     *  Locate a graphical symbol which will contain an element file name.
     */

    while ( (rc = fscanf(fp,"%d",&symbol_type)) != EOF )
        {

        /*
         *  If we don't get an integer, dump the rest of the line.  If we
         *  get a LINE SEGMENT indicator, we are finished with the file
         *  because the line info follows the symbol info.
         */

        if ( rc == 0 )
            {
            fscanf( fp, "%*[^\n]" );
            continue;
            }

        if ( symbol_type == SEGMENT_KEY )
            break;
```

```
        /*
         *  See if we have a symbol which may contain a reference to an
         *  element file.
         */

        switch ( symbol_type )
            {
            case START:
            case STOP:
            case GOTO:  fscanf( fp,"%*u %*d %*d %*d %*d %*d %*d %*d %*d %*d %*d" );
                        read_element_str( fp, elementName );
                        fscanf( fp, "%*u %*u %*d" );
                        fscanf( fp, "%*[^\n]" );
                        break;
            default :   fscanf( fp, "%*[^\n]" );
                        continue;
            }

        /*
         *  If the name in the symbol matches the name of the Element we are working
         *  on, then bump the counter.
         */

        if (! strcmp(el_file,elementName) )
            count++;
        }

    /*
     *  Close the element file.
     */

    fclose( fp );
    return( count );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  implode()
*
* MODULE FUNCTION:
*
*   This routine loads the element to be imploded into.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  implode( element_name, type )

    char    *element_name;
    int     type;

{
    /*
     *  Save the current element if needed.
     */

    if ( SaveNeeded )
        save_curr_element();

    /*
     *  Init element variables.
     */

    reinit_element_vars();

    if ( type == ELEMENT )
        {
        strcpy( GElementFile, element_name );
        strcat( GElementFile, EL_EXT );
        ElementType = ELEMENT;
        }
    else
        {
        strcpy( GElementFile, LibPath );
        strcat( GElementFile, "/" );
        strcat( GElementFile, element_name );
        strcat( GElementFile, EL_EXT );
        ElementType = LIB;
        }

    strcpy( ElementFile, element_name );

    /*
     *  See if the element file exists, if it doesn't ask the user if they want
     *  to create one.
     */

    if ( access(GElementFile,R_OK) )
        if ( ask("Selected Element file does not exist, create one?") )
            {
               arm_tgl( tgl_cre_elem_ce );
            disarm_tgl( tgl_cre_elem_le );
            XmTextSetString( txt_cre_elem_ne, ElementFile );
```

```
        XtManageChild( dlg_cre_elem );
        busy( dlg_cre_elem, FALSE );
        }
    else
        {
        user_ack("Element file not created, please select another");
        ValidElement = False;
        busy( dlg_sel_elem, FALSE );
        return;
        }

/*
 *  File exists, read it.
 */

else
    if ( read_element_file() == ERR )
        {
        user_ack("Unable to read Element file - please check error log");
        elog(1,"Unable to read Element file - %s", GElementFile );
        }

upd_pos_panel( ELEMENT_PURPOSE );
upd_mode_panel();
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  init_element_vars()
*
* MODULE FUNCTION:
*
*   This routine initializes the Element variables.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  init_element_vars()
{
    int  i, value, slider, incr, page_incr;
    Widget  hscroll, vscroll;
    Arg     args[2];


    Audit       = FALSE;
    ElementType = -1;
    LineSegKey  = 1;
    LineKey     = 1;
    LineListKey = 1;
    SaveNeeded  = FALSE;
    Snap        = FALSE;
    SymKey      = 1;


    ElementFile[0]  = NULL;
    GElementFile[0] = NULL;

    CreateDate[0]   = NULL;
    UpdateDate[0]   = NULL;
    UpdateTime[0]   = NULL;

    ValidElement    = FALSE;
    nextsymbol      = 0;

    strcpy( Author, UserName );

    /*
     *  The entire textsw widget must be cleared or the end of the last
     *  string will be visible when the purpose is updated.
     */

    for ( i=0; i<MAX_PURPOSE; i++ )
        ElementPurpose[i] = ' ';
    ElementPurpose[0]     = NULL;

    for ( i=0; i<MAX_SYMBOLS; i++ )
        if (symbols[i])
            {
            XtUnrealizeWidget( symbols[i] );
              XtDestroyWidget( symbols[i] );
            symbols[i] = NULL;
            }

    XClearArea( display, XtWindow(draw_area), 0,0,0,0, True );
```

```
    XSetFunction( display, DefaultGC(display,DefaultScreen(display)), GXcopy );

    /*
     *  Initialize the symbol array and Work Area cell map.
     */

    init_symbol_array();
    init_cell_map();

    set_null_undo_event();

    Audited = 0;

    /*
     *  Reset the Work Area scroll bars...
     */

    /*
     *  Get the work area's scrollbars
     */

    XtSetArg( args[0], XmNhorizontalScrollBar,   &hscroll );
    XtSetArg( args[1], XmNverticalScrollBar,     &vscroll );
    XtGetValues( scr_WA, args, 2 );

    /*
     *  Get the current scrollbar values
     */

    XmScrollBarGetValues( hscroll, &value, &slider, &incr, &page_incr, True );

    /*
     *  Change only the current scroll bar position. The True parameter
     *  indicates that the ChangedPosition callback should be called.
     */

    XmScrollBarSetValues( hscroll, 0, slider, incr, page_incr, True );
    XmScrollBarSetValues( vscroll, 0, slider, incr, page_incr, True );

    expr_text       = NULL;
    sym_text        = NULL;
    comment_text    = NULL;

    /*
     *  Start in zoomed in mode.
     */

    Zoomed = FALSE;
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  load_element_list()
*
* MODULE FUNCTION:
*
*   This routine loads the Element file names from the Comp file into the element
*   selection list widget.  This routine also loads the Comp names into the Comp
*   selection list widget during ACTIVATE and STOPs.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  load_element_list( type, list_widget )

    Widget      list_widget;
    int         type;

{
    DIR             *dir;
    FILE            *fp;
    XmString        xmstr;
    struct dirent   *dp;
    int             i,
                    cnt=0;
    char            *cptr,
                    *nptr;


    /*
     *  Need a list of Comps for an ACTIVATE or STOP list.
     */

    if ( type == COMP )
        {
        if ( ! ValidPosition )
            {
            user_ack("Please select a Position first");
            return( ERR );
            }

        /*
         *  Delete the list entries.
         */

        XmListDeleteAllItems( list_widget );

        /*
         *  Open the Position directory and list get all the Comp names.
         */

        if ((dir = opendir(PositionPath)) == NULL)
            {
            elog(1,"load_element_list() - couldn't read Position directory");
            return( ERR );
            }

        for ( cnt=0, dp=readdir(dir); dp!=NULL; dp=readdir(dir) )
```

```
        {
        strcpy( selList[cnt], dp->d_name );

        /*
         *  If we find a valid Comp dir name, remove the extension from the
         *  string which will be inserted into the selection list widget.
         */

        if (! valid_comp_dir_name(selList[cnt]) )
            {
            nptr =  selList[cnt];
            cptr = &selList[cnt][strlen(nptr)-4];
            *cptr = NULL;
            cnt++;
            }

        if ( cnt == MAX_FILES )
            {
            user_ack("Error: exhausted file list - notify developer");
            elog(1,"load_element_list() - exhausted file list");
            return( ERR );
            }
        }

    /*
     *  Insert the Comp names into the selection list widget.
     */

    if ( cnt > 0 )
        {
        qsort( selList, cnt, MAX_NAME, strcmp );
        for ( i=0; i<cnt; i++ )
            {
            xmstr = XmStringCreate( selList[i], XmSTRING_DEFAULT_CHARSET );
            XmListAddItem( list_widget, xmstr, i+1 );
            XmStringFree( xmstr );
            }
        }

    return( OK );
    }

/*
 *  Need a list of Element names.  Check to make sure we have a Comp File.
 */

if (! ValidComp )
    if ( ask("No Comp selected, do you wish to edit orphan Library elements?") )
        return( OK );
    else
        {
        user_ack("Please select a Comp, then select one of the Comp's elements");
        return( ERR );
        }

/*
 *  Open the comp file and load the list.
 */

if (! (fp = fopen(GCompFile,"r")) )
    {
    user_ack("Error: could not open comp file");
    elog(1,"load_element_list: couldn't open comp file: %s",GCompFile);
    return( ERR );
    }

if ((cnt = read_comp_list(fp,type,selList)) == ERR)
    {
    user_ack("Error: could not read comp file");
    elog(1,"load_element_list: couldn't read comp list of file: %s",GCompFile);
    fclose( fp );
    return( ERR );
    }

fclose( fp );

/*
 *  Mark the first entry as the root element.
 */

if ( type == ELEMENT )
    strcat( selList[0], "          root Element" );

/*
 *  Delete the list entries.
 */

XmListDeleteAllItems( list_widget );

/*
 *  Sort the list and install the list into the selection widget.
 */

if ( cnt > 0 )
    {
    qsort( selList, cnt, MAX_NAME, strcmp );
    for ( i=0; i<cnt; i++ )
        {
        xmstr = XmStringCreate( selList[i], XmSTRING_DEFAULT_CHARSET );
        XmListAddItem( list_widget, xmstr, i+1 );
        XmStringFree( xmstr );
        }
    }

return( OK );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  locate_line()
*
* MODULE FUNCTION:
*
*   This routine is used to locate a line structure during the reading of an Element
*   file from disk.  The proper line structure is identified by the key which was
*   used when the line information was written to disk.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

Line  *locate_line( key )

    unsigned int key;

{
    struct readLine *linePtr;


    if ( key == 0 )
        return( NULL );

    linePtr = ReadLineRoot;

    /*
     *  Locate the Line structure in the linked list.
     */

    while ( linePtr != NULL )
        {
        if ( linePtr->key == key )
            return( linePtr->line );
        linePtr = linePtr->next;
        }

    elog(1,"locate_line() - serious internal error - couldn't find Line struct %u",key);
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  locate_list()
*
* MODULE FUNCTION:
*
*   This routine is used to locate a line list structure during the reading of an
*   Element file from disk.  The proper line list structure is identified by the key
*   which was used when the line information was written to disk.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

LineList  *locate_list( key )

    unsigned int key;

{
    struct readLineList *listPtr;


    if ( key == 0 )
        return( NULL );

    listPtr = ReadLineListRoot;

    /*
     *  Locate the LineList structure in the linked list.
     */

    while ( listPtr != NULL )
        {
        if ( listPtr->key == key )
            return( listPtr->lineList );
        listPtr = listPtr->next;
        }

    elog(1,"locate_list() - serious error - couldn't find LineList struct %u",key);
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  locate_segment()
*
* MODULE FUNCTION:
*
*   This routine is used to locate a line segment structure during the reading of an
*   Element file from disk.  The proper line segment structure is identified by the key
*   which was used when the line information was written to disk.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

LineSeg  *locate_segment( key )

    unsigned int key;

{
    struct readLineSeg *segPtr;


    if ( key == 0 )
        return( NULL );

    segPtr = ReadLineSegRoot;

    /*
     *  Locate the LineSegment structure in the linked list.
     */

    while ( segPtr != NULL )
        {
        if ( segPtr->key == key )
            return( segPtr->lineSeg );
        segPtr = segPtr->next;
        }

   elog(1,"locate_segment() - serious error, couldn't find LineSeg struct %u",key);
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  locate_symbol()
*
* MODULE FUNCTION:
*
*   This routine is used to locate a symbol structure during the reading of an
*   Element file from disk.  The proper symbol structure is identified by the key
*   which was used when the Element file's information was written to disk.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

Symbol  *locate_symbol( key )

    unsigned int key;

{
    int sym = 0;


    if ( key == 0 )
        return( NULL );

    while( process_lines[sym].sym )
        if ( process_lines[sym].sym->key == key )
            return( process_lines[sym].sym );
        else
            sym++;

   elog(1,"locate_symbol() - serious internal error - couldn't find Symbol struct");
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  read_element_file()
*
* MODULE FUNCTION:
*
*   This routine opens and reads an Element file from disk.  This routine reconstructs
*   all the structures kept in memory to record line and symbol information.  This
*   routine also reconstructs all of the pointers in these structures.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  read_element_file()
{
    FILE            *fp;
    Line            *linePtr;
    LineList        *listPtr;
    LineSeg         *segPtr;
    Symbol          *symbol;
    char            temp_str[MAX_TEXT];
    int             attributes = 0,
                    font_size,
                    j,
                    n,
                    rc,
                    sym,
                    symbol_type;

    unsigned int    line_from,
                    line_next,
                    line_type;
    unsigned long   fore,
                    back;
    Arg             args[10];
    Pixmap          pixmap;


    ValidElement = FALSE;

    /*
     *  Try to open the element file.
     */

    if (! (fp = fopen(GElementFile,"r")) )
        {
        user_ack("Error: could not read element file");
        return( NOT_FOUND );
        }

    /*
     *  Initialize the process_lines array.
     */

    for ( sym=0; sym<MAX_SYMBOLS; sym++ )
        process_lines[sym].sym = NULL;

    /*
     *  Read the header.
     */

    if ( fscanf(fp,"%d ",&ElementType) == EOF )
        return( ERR );

    switch ( ElementType )
        {
        case LIB     :
        case ELEMENT : break;
        default      : elog(1,"read_element_file() - invalid comp type %d",ElementType);
                       return( ERR );
        }

    if ( fscanf(fp,"%s",CreateDate) == EOF )
        return( ERR );

    if ( fscanf(fp,"%s",UpdateDate) == EOF )
        return( ERR );

    if ( fscanf(fp,"%s",UpdateTime) == EOF )
        return( ERR );

    if ( read_element_str(fp,Author) )
        return( ERR );

    /*
     *  Read this element's purpose in life.
     */

    if ( read_element_str(fp,ElementPurpose) )
        return( ERR );

    /*
     *  Read the palette items' back and foreground colors
     *  and if we're on a color display, refill the palette items
     */

    for ( j=0; j < NUM_PALETTE; j++ )
        {
        fscanf( fp, "%lu %lu", &fore, &back);
        if ( Color )
            {
            set_my_foreground( j, fore );
            set_my_background( j, back );
            pixmap = XmGetPixmap( XtScreen(rc_palette), palette_items[j], colors[fore],
                                  colors[back] );
            }
        else
            {

            /*DEBUG
             *
             *  Set mono foreground and background.
             */

            set_my_foreground( j, 1 );
            set_my_background( j, 0 );
            pixmap = XmGetPixmap( XtScreen(rc_palette), palette_items[j], 1, 0 );
            }
        XtSetArg( args[0], XmNlabelPixmap, pixmap );
        XtSetValues( Palette[j], args, 1 );
        }
```

```
    /*
     *  Eat the last blank, the #, and \n on the palette colors line
     */

    fscanf( fp, "%*c%*c\n" );

    /*
     *  make entry in symbol table for this element;
     *  all of this element's vars will depend on this name.
     */

    if (! (struct symbol_entry *) lookup_symbol(NULL, ElementFile) )
        {
        /*DEBUG*/
        elog(3,"sel elem: adding entry %s to symtable", ElementFile);
        attributes |= PROCEDURE;
        if ( add_symbol_entry(NULL,ElementFile,attributes,0,0,0,0,0) )
            error_handler( ERR_ADD_SYMBOL, "read_element_file" );
        }
    else
        /*DEBUG*/
        elog(3,"sel_elem: entry %s found in symtable, not adding", ElementFile);

    /*
     *  Read the symbols.
     */

    while ( TRUE )
        {

        /*
         *  Make sure we can add more symbols.
         */

        if ( (sym = next_avail_sym()) < 0 )
            {
            user_ack("File contains too many symbols, aborting read of Element file");
            elog(1,"File contains too many symbols, aborting read of Element file");
            return( ERR );
            }

        rc = fscanf( fp, "%d", &symbol_type );
        if ((rc == EOF) || (rc == 0))
            break;

        /*
         *  Check for the start of the line information.
         */

        if ( symbol_type == SEGMENT_KEY )
            break;

        /*
         *  We have a symbol.  Read its info.
         */

        Symbol_Map[sym].symbol_type = symbol_type;

        fscanf(fp,"%u %d %d %d %d %d %d %d %d %d ",
                    &Symbol_Map[sym].key,
                    &Symbol_Map[sym].symbol_generated,
                    &Symbol_Map[sym].cell_height,
                    &Symbol_Map[sym].cell_width,
                    &Symbol_Map[sym].cell_x,
                    &Symbol_Map[sym].cell_y,
                    &Symbol_Map[sym].height,
                    &Symbol_Map[sym].width,
                    &Symbol_Map[sym].ulcx,
                    &Symbol_Map[sym].ulcy);

        fscanf(fp, "%d ", &font_size);

        if ( font_size == 0 )
            Symbol_Map[sym].font = teeny_font;
        else if ( font_size == 1 )
            Symbol_Map[sym].font = small_font;
        else if ( font_size == 2 )
            Symbol_Map[sym].font = big_font;


        /*
         *  if this is the begin symbol, set BeginSym
         */

        if ( symbol_type == BEGIN )
            Begin_Sym = &Symbol_Map[sym];

        sym_text = NULL;
        read_element_str( fp, temp_str );
        if ( *temp_str != NULL )
            {
            Symbol_Map[sym].text = malloc( strlen(temp_str)+1 );
            strcpy( Symbol_Map[sym].text, temp_str );
            sym_text = Symbol_Map[sym].text;
            }
        else
            Symbol_Map[sym].text = NULL;

        fscanf( fp,"%u %u", &line_next, &line_from );

        /*
         *  Register the symbol's line pointer keys.
         */

        process_lines[sym].sym   = &(Symbol_Map[sym]);
        process_lines[sym].next  = line_next;
        process_lines[sym].from  = line_from;
        process_lines[sym].true  = 0;
        process_lines[sym].false = 0;

        Symbol_Map[sym].next = NULL;
        Symbol_Map[sym].from = NULL;

        /*
         *
         */

        if ( Symbol_Map[sym].key > SymKey )
            SymKey = Symbol_Map[sym].key;

        /*
         *  Insert the symbol specific info.
         */

        if ( symbol_type == IF)

            /*
             *  Insert true and false line info
```

```
         */

        {
        fscanf( fp, "%u %u %d %d %d %d",
            &process_lines[sym].false,
            &process_lines[sym].true,
            &Symbol_Map[sym].Sym.IfSym.false_x,
            &Symbol_Map[sym].Sym.IfSym.false_y,
            &Symbol_Map[sym].Sym.IfSym.true_x,
            &Symbol_Map[sym].Sym.IfSym.true_y );

        /*
         *  Draw the line TRUE/FALSE labels.
         */

        if ( process_lines[sym].false )
            XDrawString( display, XtWindow(draw_area), LDgc,
                         Symbol_Map[sym].Sym.IfSym.false_x,
                         Symbol_Map[sym].Sym.IfSym.false_y,
                         "FALSE", 5 );

        if ( process_lines[sym].true )
            XDrawString( display, XtWindow(draw_area), LDgc,
                         Symbol_Map[sym].Sym.IfSym.true_x,
                         Symbol_Map[sym].Sym.IfSym.true_y,
                         "TRUE", 4 );
        }

    /*
     *  Process the IF and SET symbol text fields.
     */

    /*
     *  set global sym_Text variable to either logical or comp text,
     *  so draw routine can use it to draw the symbol.
     */

    if ( (symbol_type == IF) || (symbol_type == SET) )
        {
        read_element_str( fp, temp_str );
        if ( *temp_str != NULL )
            {
            Symbol_Map[sym].Sym.IfSym.logical_expr = malloc( strlen(temp_str)+1 );
            strcpy( Symbol_Map[sym].Sym.IfSym.logical_expr, temp_str );
            sym_text = Symbol_Map[sym].Sym.IfSym.logical_expr;
            }
        else
            Symbol_Map[sym].Sym.IfSym.logical_expr = NULL;

        read_element_str( fp, temp_str );
        if ( *temp_str != NULL )
            {
            Symbol_Map[sym].Sym.IfSym.comp_expr = malloc( strlen(temp_str)+1 );
            strcpy( Symbol_Map[sym].Sym.IfSym.comp_expr, temp_str );
            expr_text=Symbol_Map[sym].Sym.IfSym.comp_expr;
            }
        else
            Symbol_Map[sym].Sym.IfSym.comp_expr = NULL;

        if ( (!sym_text) && (!expr_text) )
            {
            user_ack("Error: Symbol did not contain logical or comp text");
            elog(1,"read_element_file: Symbol did not contain logical or comp text")
;
            return( ERR );
            }

        read_element_str( fp, temp_str );
        if ( *temp_str != NULL )
            {
            Symbol_Map[sym].Sym.IfSym.comment = malloc( strlen(temp_str)+1 );
            strcpy( Symbol_Map[sym].Sym.IfSym.comment, temp_str );
            }
        else
            Symbol_Map[sym].Sym.IfSym.comment = NULL;

        }
    else if ( (symbol_type == GOTO) ||
              (symbol_type == STOP) ||
              (symbol_type == START) )
            {
            fscanf( fp, "%d", &Symbol_Map[sym].Sym.ElemSym.comp_type );
            }


    fscanf( fp, "\n", temp_str );

    /*
     *  Set the symbol canvas attributes.
     */

    Symbol_Map[sym].mycanvas = current_symbol;

    n = 0;
    XtSetArg( args[n], XmNwidth,     Symbol_Map[sym].width ); n++;
    XtSetArg( args[n], XmNheight,    Symbol_Map[sym].height); n++;
    XtSetArg( args[n], XmNx,         Symbol_Map[sym].ulcx  ); n++;
    XtSetArg( args[n], XmNy,         Symbol_Map[sym].ulcy  ); n++;
    XtSetArg( args[n], XmNuserData, symbol_type            ); n++;
    XtSetValues( current_symbol, args, n );

    /*
     *  Insert into the cell map.
     */

    if (!(set_cell_map_sym( SYMBOL_CELL, (Symbol *) &(Symbol_Map[sym]),
            Symbol_Map[sym].cell_x,
            Symbol_Map[sym].cell_y,
            Symbol_Map[sym].cell_width,
            Symbol_Map[sym].cell_height )))
        {
        user_ack("Error: Can't allocate cell map to place symbol");
        elog(1,"Error: Can't allocate cell to place symbol type: %d",symbol_type);
        return( ERR );
        }

    /*
     *  Draw the symbol.
     */

    draw_symbol( symbol_type, current_symbol, WAgc, Symbol_Map[sym].font );

    /*
     *  reset global text pointers
     */

    sym_text    = NULL;
    expr_text   = NULL;
```

```
    /*
     *  Turn on SHOW to make new canvas visible in work area.
     */

    XtMapWidget( current_symbol );
    }

/*
 *  Read the Line info.
 */

ReadLineRoot     = ReadLine     = NULL;
ReadLineSegRoot  = ReadLineSeg  = NULL;
ReadLineListRoot = ReadLineList = NULL;

/*
 *  Read the LineSegment info.
 */

line_type = (unsigned int) symbol_type;
while ( line_type == SEGMENT_KEY )
    {
    ReadLineSeg = (struct readLineSeg *) malloc( sizeof(struct readLineSeg) );
    segPtr      = (LineSeg *)            malloc( sizeof(LineSeg) );
    fscanf( fp, "%u %u %u %d %d %d %d %d %d %d %d %d %d\n",
        &segPtr->key,
        &ReadLineSeg->next_seg,
        &ReadLineSeg->prev_seg,
        &segPtr->cell_end_x,
        &segPtr->cell_end_y,
        &segPtr->cell_start_x,
        &segPtr->cell_start_y,
        &segPtr->end_x,
        &segPtr->end_y,
        &segPtr->start_x,
        &segPtr->start_y,
        &segPtr->arrow_x,
        &segPtr->arrow_y,
        &segPtr->orientation );

    ReadLineSeg->key        = segPtr->key;
    ReadLineSeg->lineSeg    = segPtr;
    ReadLineSeg->next       = ReadLineSegRoot;
    ReadLineSegRoot         = ReadLineSeg;

    if ( ReadLineSeg->key > LineSegKey )
        LineSegKey = ReadLineSeg->key;

    LDstartX = segPtr->start_x;
    LDstartY = segPtr->start_y;
    LDendX   = segPtr->end_x;
    LDendY   = segPtr->end_y;
    set_line();

    rc = fscanf( fp, "%d ", &line_type );
    if ((rc == EOF) || (rc == 0))
        break;
    }

/*
 *  Read the Line structure info.
 */
while ( line_type == LINE_KEY )
    {
    ReadLine = (struct readLine *) malloc( sizeof(struct readLine) );
    linePtr  = (Line * )           malloc( sizeof(Line) );
    fscanf( fp, "%u %u %u %u\n",
        &linePtr->key,
        &ReadLine->line_seg,
        &ReadLine->from_sym,
        &ReadLine->to_sym );

    ReadLine->key   = linePtr->key;
    ReadLine->line  = linePtr;
    ReadLine->next  = ReadLineRoot;
    ReadLineRoot    = ReadLine;

    if ( ReadLine->key > LineKey )
        LineKey = ReadLine->key;

    rc = fscanf( fp, "%d ", &line_type );
    if ((rc == EOF) || (rc == 0))
        break;
    }

/*
 *  Read the LineList info.
 */

while ( line_type == LIST_KEY )
    {
    ReadLineList = (struct readLineList *) malloc( sizeof(struct readLineList) );
    listPtr      = (LineList *)            malloc( sizeof(Line) );
    fscanf( fp, "%u %u %u %u\n",
        &listPtr->key,
        &ReadLineList->line,
        &ReadLineList->next_list,
        &ReadLineList->prev_list );

    ReadLineList->key       = listPtr->key;
    ReadLineList->lineList  = listPtr;
    ReadLineList->next      = ReadLineListRoot;
    ReadLineListRoot        = ReadLineList;

    if ( ReadLineList->key > LineListKey )
        LineListKey = ReadLineList->key;

    rc = fscanf( fp, "%d ", &line_type );
    if ((rc == EOF) || (rc == 0))
        break;
    }

/*
 *  File has been successfully read.
 */

fclose( fp );

/*
 *  Go through the process_lines array and set the pointers in the
 *  symbols to the newly created line structures.
 */

sym = 0;
while( process_lines[sym].sym )
    {
```

```
    symbol = process_lines[sym].sym;
    symbol->next = locate_line( process_lines[sym].next );
    symbol->from = locate_list( process_lines[sym].from );

    if ( symbol->symbol_type == IF )
        {
        symbol->Sym.IfSym.false_line = locate_line( process_lines[sym].false );
        symbol->Sym.IfSym.true_line  = locate_line( process_lines[sym].true );
        }
    sym++;
    }

/*
 *  Go through all the Line Segments and update their pointers to the sibling
 *  line segments.
 */

ReadLineSeg = ReadLineSegRoot;
while ( ReadLineSeg != NULL )
    {
    segPtr = ReadLineSeg->lineSeg;
    segPtr->next = (struct LineSeg *) locate_segment(ReadLineSeg->next_seg);
    segPtr->prev = (struct LineSeg *) locate_segment(ReadLineSeg->prev_seg);

    if (!(segPtr->next))

        /*
         *  last seg in a line - draw arrow
         */

        draw_arrows(segPtr, FALSE);

    ReadLineSeg = ReadLineSeg->next;
    }

/*
 *  Go through all the Line Structures and update their pointers.
 */

ReadLine = ReadLineRoot;
while ( ReadLine != NULL )
    {
    linePtr       = ReadLine->line;
    linePtr->from = (struct Symbol *) locate_symbol( ReadLine->from_sym );
    linePtr->to   = (struct Symbol *) locate_symbol( ReadLine->to_sym );
    linePtr->line = locate_segment( ReadLine->line_seg );
    segPtr = linePtr->line;
    while ( segPtr != NULL )
        {
        set_cell_map_line( segPtr, linePtr );
        segPtr = (LineSeg *) segPtr->next;
        }
    ReadLine = ReadLine->next;
    }

/*
 *  Go through all the LineList structures and update their pointers.
 */

ReadLineList = ReadLineListRoot;
while( ReadLineList != NULL )
    {
    listPtr = ReadLineList->lineList;
    listPtr->line = locate_line( ReadLineList->line );
    listPtr->next = (struct LineList *) locate_list( ReadLineList->next_list );
    listPtr->prev = (struct LineList *) locate_list( ReadLineList->prev_list );
    ReadLineList = ReadLineList->next;
    }

/*
 *  Free all the temporary line structures.
 */

ReadLine = ReadLineRoot;
while ( ReadLine != NULL )
    {
    readTempLine = ReadLine;
    ReadLine     = ReadLine->next;
    free( readTempLine );
    }

ReadLineSeg = ReadLineSegRoot;
while ( ReadLineSeg != NULL )
    {
    readTempSeg = ReadLineSeg;
    ReadLineSeg = ReadLineSeg->next;
    free( readTempSeg );
    }

ReadLineList = ReadLineListRoot;
while ( ReadLineList != NULL )
    {
    readTempList = ReadLineList;
    ReadLineList = ReadLineList->next;
    free( readTempList );
    }

/*
 *  Bump the pointer/symbol keys to a new, unique value.
 */

SymKey++;
LineKey++;
LineSegKey++;
LineListKey++;

ValidElement = TRUE;

/*
 *  removing this symbol may change element status to complete
 */

if ( complete(0, LINES_AND_EXPR) == ERR )
    upd_status( 1 );
else
    upd_status( 0 );

return( OK );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  read_element_str()
*
* MODULE FUNCTION:
*
*   This routine is used to read character strings from Element files.  A count of
*   the number of characters in the string is read first, and then the specified
*   number of characters are read into the string.  This method was used so strings
*   containing spaces could be written and then read from Element files.  The strings
*   which this routine read should have been written using save_element_str().
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  read_element_str( fp, string )

    FILE *fp;
    char *string;

{
    int length;

    /*
     *  Read an integer number of characters which indicate the length of this
     *  string.
     */

    if ( fscanf(fp,"%d ",&length) == EOF )
        return( ERR );

    /*
     *  If the string is not zero length and if the user has supplied a valid
     *  pointer, then read the specified number of characters into the str pointer
     *  supplied.
     */

    if ( length )
        {
        if ( string )
            {
            if ( fread(string,length,1,fp) != 1 )
                return( ERR );
            string[length] = NULL;
            }
        else

            /*
             *  The caller did not supply a pointer, must be they don't want the
             *  data, skip past the string.
             */

            if ( fseek(fp,length,1) )
                return( ERR );
        }
    else if ( string )
        *string = NULL;

    return( OK );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  record_line()
*
* MODULE FUNCTION:
*
*   This routine is used to build line structures.  These line structures will be
*   used to record line information during the saving of Element files to disk.
*   These structures contain integer keys instead of pointers.  These keys will be
*   used to reconstruct the pointers when the Element file is read from disk.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  record_line( line )

    Line *line;

{
    struct saveLine *tempLine;
    Symbol          *sym;


    SaveLine = SaveLineRoot;

    /*
     *  See if this Line structure is already in the linked list.
     */

    while ( SaveLine != NULL )
        {
        if ( SaveLine->key == line->key )
            return;
        SaveLine = SaveLine->next;
        }

    /*
     *  The current Line structure was not found in the linked list.  Add the
     *  Line strucutre info to the linked list.
     */

   tempLine = (struct saveLine *) malloc( sizeof(struct saveLine) );
   tempLine->key       = line->key;
   tempLine->line_seg  = line->line->key;

   sym = (Symbol *) line->from;
   tempLine->from      = sym->key;

   sym = (Symbol *) line->to;
   tempLine->to        = sym->key;

   tempLine->next      = SaveLineRoot;
   SaveLineRoot        = tempLine;

   /*
    *  Record this Line structure's line segements.
    */

    record_segment( line->line );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  record_list()
*
* MODULE FUNCTION:
*
*   This routine is used to build line list structures.  These line list structures will
*   be used to record line information during the saving of Element files to disk.
*   These structures contain integer keys instead of pointers.  These keys will be
*   used to reconstruct the pointers when the Element file is read from disk.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  record_list( list )

    LineList *list;

{
    struct saveLineList *tempList;
    LineList *nextList;


    SaveLineList = SaveLineListRoot;

    /*
     *  See if this LineList structure is already in the linked list.
     */

    while ( SaveLineList != NULL )
        {
        if ( SaveLineList->key == list->key )
            return;
        SaveLineList = SaveLineList->next;
        }

    /*
     *  The current LineList structure was not found in the linked list.  Add
     *  the LineList strucutre info to the linked list.
     */

    tempList = (struct saveLineList *) malloc( sizeof(struct saveLineList) );
    tempList->key       = list->key;
    tempList->line      = list->line->key;

    if ( list->next )
        {
        nextList = (LineList *) list->next;
        tempList->next_list = nextList->key;
        }
    else
        tempList->next_list = 0;

    if ( list->prev )
        {
        nextList = (LineList *) list->prev;
        tempList->prev_list = nextList->key;
        }
    else
        tempList->prev_list = 0;

    tempList->next      = SaveLineListRoot;
    SaveLineListRoot    = tempList;

    /*
     *  Record this LineList structure's Line structure.
     */

    record_line( list->line );

    /*
     *  Save the other LineLists which enter this symbol.
     */

    if ( list->next )
        record_list( (LineList *) list->next );
    if ( list->prev )
        record_list( (LineList *) list->prev );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  record_segment()
*
* MODULE FUNCTION:
*
*   This routine is used to build line segment structures.  These line segment
*   structures will be used to record line information during the saving of Element
*   files to disk.  These structures contain integer keys instead of pointers.  These
*   keys will be used to reconstruct the pointers when the Element file is read from
*   disk.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  record_segment( line_seg )

    LineSeg *line_seg;

{
    struct saveLineSeg *tempSeg;
    LineSeg *nextSeg;


    SaveLineSeg = SaveLineSegRoot;

    /*
     *  See if this Line segment is already in the linked list.
     */

    while ( SaveLineSeg != NULL )
        {
        if ( SaveLineSeg->key == line_seg->key )
            return;
        SaveLineSeg = SaveLineSeg->next;
        }

    /*
     *  The current Line segment was not found in the linked list.  Add the
     *  Line segment info to the linked list.
     */

    tempSeg = (struct saveLineSeg *) malloc( sizeof(struct saveLineSeg) );
    tempSeg->key = line_seg->key;

    if ( line_seg->next )
        {
        nextSeg = (LineSeg *) line_seg->next;
        tempSeg->next_seg   = nextSeg->key;
        }
    else
        tempSeg->next_seg   = 0;

    if ( line_seg->prev )
        {
        nextSeg = (LineSeg *) line_seg->prev;
        tempSeg->prev_seg   = nextSeg->key;
        }
    else
        tempSeg->prev_seg   = 0;

    tempSeg->cell_end_x     = line_seg->cell_end_x;
    tempSeg->cell_end_y     = line_seg->cell_end_y;
    tempSeg->cell_start_x   = line_seg->cell_start_x;
    tempSeg->cell_start_y   = line_seg->cell_start_y;
    tempSeg->end_x          = line_seg->end_x;
    tempSeg->end_y          = line_seg->end_y;
    tempSeg->start_x        = line_seg->start_x;
    tempSeg->start_y        = line_seg->start_y;
    tempSeg->arrow_x        = line_seg->arrow_x;
    tempSeg->arrow_y        = line_seg->arrow_y;
    tempSeg->orientation    = line_seg->orientation;

    tempSeg->next           = SaveLineSegRoot;
    SaveLineSegRoot         = tempSeg;

    /*
     *  Record the other segments which make up this line.
     */

    if ( line_seg->next )
        record_segment( (LineSeg *) line_seg->next );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  reinit_element_vars()
*
* MODULE FUNCTION:
*
*   This routine is used to clear/free Element file information when the user changes
*   Element files.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  reinit_element_vars()
{
    free_cellmap_symbols();
    free_cellmap_linelists();
    free_lines();

    init_element_vars();

    upd_pos_panel( NO_CHANGE );

    upd_mode_panel();
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  save_curr_element()
*
* MODULE FUNCTION:
*
*   This routine is called when the user selects Save Element.  This routine will
*   make sure the Element needs to be saved, and then will call the proper routine
*   to save the Element file.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  save_curr_element()
{
    if ( SaveNeeded && (! ValidElement) )
        {
        user_ack("Can not save changes, a valid Element file was not created/selected");
        return;
        }

    if ( ask("Do you want to save your changes to the current element file?") )
        {
        save_element_file();
        if ( ValidComp )
            update_comp_file( GCompFile, CompPurpose, RootElement, NULL, NULL );
        else
            {
            user_ack("ValidComp not set during save_curr_element");
            elog(1,"ValidComp not set during save_curr_element");
            }
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  save_element_file()
*
* MODULE FUNCTION:
*
*   This routine will save the currently active Element file to disk.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.01 - 08/01/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  save_element_file()
{
    FILE    *fp;
    int     i,
            sym,
            zoomed;
    time_t  clock;


    /*
     *  If we are zoomed out, zoom back in so as to restore the correct
     *  sizes of the symbols and lines.
     */

    if ( Zoomed )
        {
        zoomed = 1;
        zoom( 1 );
        }
    else
        zoomed = 0;

    /*
     *  Try to open the element file.
     */


    /*DEBUG*/
    elog(3,"save_element_file: GElementFile is %s", GElementFile);

    if (! (fp = fopen(GElementFile,"w")) )
        {
        user_ack("Unable to open the Element file for writing");
        elog(1,"Unable to open the Element file: %s for writing",GElementFile);
        return;
        }

    /*
     *  Update the Update date field date string in the date portion of the
     *  file.
     */

    clock = time( NULL );
    strftime( UpdateDate, 11, "%m/%d/%Y", localtime(&clock) );
    strftime( UpdateTime, 9,  "%T",       localtime(&clock) );

    /*
     *  Write the header.
     */

    fprintf(fp,"%d ", ElementType);
    fprintf(fp,"%s ", CreateDate);
    fprintf(fp,"%s ", UpdateDate);
    fprintf(fp,"%s ", UpdateTime);
    save_element_str( fp, Author );

    /*
     *  Write this element's purpose in life.
     */

    save_element_str( fp, ElementPurpose );

    /*
     *  Write the palette items' back and foreground colors
     */

    for ( i=0; i < NUM_PALETTE; i++ )
        {
        fprintf( fp, "%lu ", get_my_foreground(i) );
        fprintf( fp, "%lu ", get_my_background(i) );
        }

    fprintf( fp, "#\n" );

    /*
     *  Initialize the save line info pointers.
     */

    SaveLineRoot        = NULL;
    SaveLineListRoot    = NULL;
    SaveLineSegRoot     = NULL;

    SaveLine            = NULL;
    SaveLineList        = NULL;
    SaveLineSeg         = NULL;

    /*
     *  Save the symbols.
     */

    for ( sym=0; sym<MAX_SYMBOLS; sym++ )
        {

        /*
         *  Make sure this symbol map entry is active.
         */

        if ( Symbol_Map[sym].symbol_type != NONE )
            {
            fprintf( fp, "%d %u %d %d %d %d %d %d %d %d %d ",
                         Symbol_Map[sym].symbol_type,
                         Symbol_Map[sym].key,
                         Symbol_Map[sym].symbol_generated,
                         Symbol_Map[sym].cell_height,
                         Symbol_Map[sym].cell_width,
                         Symbol_Map[sym].cell_x,
                         Symbol_Map[sym].cell_y,
                         Symbol_Map[sym].height,
                         Symbol_Map[sym].width,
                         Symbol_Map[sym].ulcx,
```

```
                Symbol_Map[sym].ulcy);

if ( Symbol_Map[sym].font == teeny_font )
    fprintf( fp, "%d ", 0);
else if ( Symbol_Map[sym].font == small_font )
    fprintf( fp, "%d ", 1);
else if ( Symbol_Map[sym].font == big_font )
    fprintf( fp, "%d ", 2);

save_element_str( fp, Symbol_Map[sym].text );

/*
 *  Save the "next" pointer.
 */

if ( Symbol_Map[sym].next != NULL )
    {
    fprintf( fp, " %u ", Symbol_Map[sym].next->key );
    record_line( Symbol_Map[sym].next );
    }
else
    fprintf( fp, " %u ", 0 );

/*
 *  Save the "from" pointer.
 */

if ( Symbol_Map[sym].from != NULL )
    {
    fprintf( fp, "%u ", Symbol_Map[sym].from->key );
    record_list( Symbol_Map[sym].from );
    }
else
    fprintf( fp, "%u ", 0 );

/*
 *  Depending on the symbol type, save the other fields.
 */

if ( Symbol_Map[sym].symbol_type == IF)
    {

    /*
     *  Save the IF symbol's TRUE and FALSE line pointers.
     */

    if ( Symbol_Map[sym].Sym.IfSym.false_line != NULL )
        {
        fprintf( fp, "%u ", Symbol_Map[sym].Sym.IfSym.false_line->key );
        record_line( Symbol_Map[sym].Sym.IfSym.false_line );
        }
    else
        fprintf( fp, "%u ", 0 );

    if ( Symbol_Map[sym].Sym.IfSym.true_line != NULL )
        {
        fprintf( fp, "%u ", Symbol_Map[sym].Sym.IfSym.true_line->key );
        record_line( Symbol_Map[sym].Sym.IfSym.true_line );
        }
    else
        fprintf( fp, "%u ", 0 );

    fprintf( fp, "%d %d %d %d ",
        Symbol_Map[sym].Sym.IfSym.false_x,
        Symbol_Map[sym].Sym.IfSym.false_y,
        Symbol_Map[sym].Sym.IfSym.true_x,
        Symbol_Map[sym].Sym.IfSym.true_y );

    /*
     *  Save the text fields.
     */

    save_element_str( fp, Symbol_Map[sym].Sym.IfSym.logical_expr );
    save_element_str( fp, Symbol_Map[sym].Sym.IfSym.comp_expr );
    save_element_str( fp, Symbol_Map[sym].Sym.IfSym.comment );
    }

/*
 *  Save the SET symbol's specific fields.
 */

else if ( Symbol_Map[sym].symbol_type == SET)
    {
    save_element_str( fp, Symbol_Map[sym].Sym.IfSym.logical_expr );
    save_element_str( fp, Symbol_Map[sym].Sym.IfSym.comp_expr );
    save_element_str( fp, Symbol_Map[sym].Sym.IfSym.comment );
    }
else if ((Symbol_Map[sym].symbol_type == GOTO) ||
         (Symbol_Map[sym].symbol_type == STOP) ||
         (Symbol_Map[sym].symbol_type == START) )
    {
    fprintf( fp, "%d", Symbol_Map[sym].Sym.ElemSym.comp_type );
    }


fprintf( fp, "\n" );
}

/*
 *  Save the line information that was generated while saving the symbols.
 *
 *  Save the list of line segments.  Free the line segment structure
 *  after writing its contents to the file.
 */

SaveLineSeg = SaveLineSegRoot;
while( SaveLineSeg != NULL )
    {
    fprintf( fp, "%d ", SEGMENT_KEY );
    fprintf( fp, "%u %u %u %d %d %d %d %d %d %d %d %d %d %d\n",
        SaveLineSeg->key,
        SaveLineSeg->next_seg,
        SaveLineSeg->prev_seg,
        SaveLineSeg->cell_end_x,
        SaveLineSeg->cell_end_y,
        SaveLineSeg->cell_start_x,
        SaveLineSeg->cell_start_y,
        SaveLineSeg->end_x,
        SaveLineSeg->end_y,
        SaveLineSeg->start_x,
        SaveLineSeg->start_y,
        SaveLineSeg->arrow_x,
        SaveLineSeg->arrow_y,
        SaveLineSeg->orientation );
    tempLineSeg = SaveLineSeg;
    SaveLineSeg = SaveLineSeg->next;
    free( tempLineSeg );
```

```
    }

/*
 *  Save the list of Lines.
 */

SaveLine = SaveLineRoot;
while ( SaveLine != NULL )
    {
    fprintf( fp, "%d ", LINE_KEY );
    fprintf( fp, "%u %u %u %u\n",
        SaveLine->key,
        SaveLine->line_seg,
        SaveLine->from,
        SaveLine->to );
    tempLine = SaveLine;
    SaveLine = SaveLine->next;
    free( tempLine );
    }

/*
 *  Save the list of LineList information.
 */

SaveLineList = SaveLineListRoot;
while ( SaveLineList != NULL )
    {
    fprintf( fp, "%d ", LIST_KEY );
    fprintf( fp, "%u %u %u %u\n",
        SaveLineList->key,
        SaveLineList->line,
        SaveLineList->next_list,
        SaveLineList->prev_list );
    tempLineList = SaveLineList;
    SaveLineList = SaveLineList->next;
    free( tempLineList );
    }

/*
 *  Close the element file.
 */

fclose( fp );

/*
 *  Mark the file as saved.
 */

upd_pos_panel( NO_CHANGE );
SaveNeeded = FALSE;

/*
 *  Update this Element's Installed flag in the symbol table so we know it
 *  needs to be reInstalled.
 */

set_sym_attribs( ElementFile, NOT_INSTALLED, False );

/*
 *  Update this element's Comp file.
 */

update_comp_file( GCompFile, NULL, NULL, NULL, NULL );

/*
 *  If we were zoomed out, zoom back out so that work area is undisturbed
 *  by save.
 */

if ( zoomed )
    zoom( 2 );

return;
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  save_element_str()
*
* MODULE FUNCTION:
*
*   This routine is used to write character strings to Element disk files.  A count of
*   the number of characters in the string is first written to the file.  This count
*   is necessary because the Element file strings often contain spaces.  The strings
*   written with this routine should be read with: read_element_str().
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  save_element_str( fp, string )

    FILE *fp;
    char *string;

{
    int length;


    if ( string != NULL )
        {
        length = strlen( string );
        fprintf( fp, "%d ", length );
        fwrite ( string, length, 1, fp );
        fprintf( fp, " " );
        }
    else
        fprintf( fp, "%d ", 0 );
}
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    element_file.h
*
* FILE FUNCTION:
*
*   This file contains the constants and variable declarations used in element_file.c
*   to maintain Element files.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /home/project/3531/Docu/GCB.spec.doc
*
*
* FILE MODULES:
*
*   N/A
*
*****************************************<----->*****************************************/

/*
 *  Constants.
 */

#define CBR_DONE        1
#define CBR_CANCEL      2
#define CBR_TYPE        3
#define CBR_DELETE      4

#define HGT_PURPOSE     100
#define MAX_FILES       150
#define WID_PURPOSE     271

#define LINES_AND_EXPR  1
#define JUST_LINES      2
#define JUST_EXPR       3

#define NO_BEGIN        -2
#define NO_END          -3


/*
 *  Structure definitions.  Most of these structures are used to build linked lists
 *  to record line information to disk files.
 */

struct proc_line
    {
    Symbol          *sym;
    unsigned int    false,
                    from,
                    next,
                    true;
    };

struct saveLine
    {
    unsigned int    key,
                    line_seg,
                    from,
                    to;

    struct saveLine *next;
    };

struct saveLineList
    {
    unsigned int    key,
                    line,
                    next_list,
                    prev_list;

    struct saveLineList *next;
    };

struct saveLineSeg
    {
    unsigned int    key,
                    next_seg,
                    prev_seg;
    int             cell_end_x,
                    cell_end_y,
                    cell_start_x,
                    cell_start_y,
                    end_x,
                    end_y,
                    start_x,
                    start_y,
                    arrow_x,
                    arrow_y,
                    orientation;

    struct saveLineSeg *next;
    };

/*
 *  Save line information linked list pointers.
 */

struct saveLine     *tempLine,
                    *SaveLine,
                    *SaveLineRoot;

struct saveLineList *tempLineList,
                    *SaveLineList,
                    *SaveLineListRoot;

struct saveLineSeg  *tempLineSeg,
                    *SaveLineSeg,
                    *SaveLineSegRoot;

/*
 *  Read line information structures.
 */

struct readLine
    {
    unsigned int    key,
                    line_seg,
                    from_sym,
                    to_sym;
    Line            *line;
    struct readLine *next;
    };

struct readLineSeg
    {
```

```
    unsigned int        key,
                        next_seg,
                        prev_seg;
    LineSeg             *lineSeg;
    struct readLineSeg  *next;
    };

struct readLineList
    {
    unsigned int        key,
                        line,
                        next_list,
                        prev_list;
    LineList            *lineList;
    struct readLineList *next;
    };

/*
 *  Read line information linked list pointers.
 */

struct readLine     *readTempLine,
                    *ReadLine,
                    *ReadLineRoot;

struct readLineSeg  *readTempSeg,
                    *ReadLineSeg,
                    *ReadLineSegRoot;

struct readLineList *readTempList,
                    *ReadLineList,
                    *ReadLineListRoot;

/*
 *  Global variables for element_file.c
 */

struct proc_line    process_lines[MAX_SYMBOLS];

extern char         *palette_items[];

char                selList[MAX_FILES][MAX_NAME];

int                 ElementListType,    /*  the type   of elements in the sel list  */
                    CallType;           /*  type of element referenced in current goto,
                                         *  start, or stop symbol                   */


/*
 *  Function prototypes.
 */

XtCallbackProc  cbr_create_elem_copy(),
                cbr_element_type(),
                cbr_el_delete(),
                cbr_el_del_sel(),
                cbr_el_selected(),
                cbr_new_element(),
                cbr_root_selected(),
                cbr_sav_element();

Line            *locate_line();
LineList        *locate_list();
LineSeg         *locate_segment();
Symbol          *locate_symbol();

int             get_element_selfcalls(),
                load_element_list(),
                read_element_file(),
                read_element_str();

void            cbr_cre_element(),
                cbr_sel_element(),
                init_element_vars(),
                record_list(),
                record_segment(),
                reinit_element_vars(),
                save_curr_element(),
                save_element_file(),
                save_element_str();
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    expr_list.c
*
* FILE FUNCTION:
*
*   Contains the routines which load the User Define Function selection list which
*   is used during the building of expressions.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /home/project/3531/Docu/GCB.spec.doc
*
*
* FILE MODULES:
*
*   load_fn_list()      - load the user defined function selection list
*   valid_fn_name()     - check for a user defined function object file name
*
*****************************************<----->*****************************************/

#include <stdio.h>
#include <dirent.h>
#include <X11/Intrinsic.h>
#include <X11/Shell.h>
#include <Xm/Xm.h>
#include "gcb.h"
#include "widgets.h"
#include "element_file.h"
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  load_fn_list()
*
* MODULE FUNCTION:
*
*   This routine creates a list of all the object files ( *.o ) in the User Defined
*   Function directory.  This routine sorts the list and then inserts the list into
*   the selection list widget.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  load_fn_list( selbox )

    Widget  selbox;

{
    DIR             *dir;
    Widget          list;
    XmString        tcs;
    struct dirent   *dp;
    int             cnt,
                    i;


    list = (Widget) XmSelectionBoxGetChild( selbox, XmDIALOG_LIST );

    /*
     *  Delete the current list entries.
     */

    XmListDeleteAllItems( list );

    /*
     *  Open the directory where the User Defined Functions are supposed to
     *  exist.
     */

    if ((dir = opendir(UserFuncsPath)) == NULL)
        {
        elog(1,"load_fn_list() - couldn't read directory");
        return( ERR );
        }

    /*
     *  Read every entry in the directory and check for object files.
     */

    for ( cnt=0, dp=readdir(dir); dp!=NULL; dp=readdir(dir) )
        {
        strcpy( selList[cnt], dp->d_name );
        if ( valid_fn_name(selList[cnt]) == OK )
            cnt++;
        if ( cnt == MAX_FILES )
            {
            user_ack("Error: exhausted file list - notify developer");
            elog(1,"load_fn_list() - exhausted file list");
```

```
            return( ERR );
            }
        }

    /*
     *  If entries exists, sort them and then insert them into the selection
     *  list.
     */

    if ( cnt > 0 )
        {
        qsort( selList, cnt, MAX_NAME, strcmp );
        for ( i=0; i<cnt; i++ )
            {
            tcs = XmStringCreate( selList[i], XmSTRING_DEFAULT_CHARSET );
            XmListAddItem( list, tcs, i+1 );
            XmStringFree( tcs );
            }
        }


    return( OK );
}
```

```
/*****************************************<----->*************************************
*
* MODULE NAME:  valid_fn_name()
*
* MODULE FUNCTION:
*
*   This routine determines if a filename is a valid User Defined Function object
*   filename.
*
*   This routine could be made simpler using strncmp(), but it would be slower than
*   using the local character pointer.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*************************************/

int  valid_fn_name( str )

    char *str;

{
    char *ptr;
    int   len;

    len = strlen( str );
    ptr = str;

    /*
     *  First make sure the name starts with a "FN_".  All user defined functions
     *  must start with a "FN_".
     */

    if ( *ptr != 'F' )
        return( ERR );
    ptr++;

    if ( *ptr != 'N' )
        return( ERR );
    ptr++;

    if ( *ptr != '_' )
        return( ERR );

    /*
     *  Now make sure the extension is ".o"
     */

    ptr = &str[len-1];
    if ( *ptr != 'o' )
        return( ERR );
    ptr--;

    if ( *ptr != '.' )
        return( ERR );

    return( OK );
}
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    expr_menu.c
*
* FILE FUNCTION:
*
*   This file contains the routines which create the math menu for if/set stmt input
*
*
* SPECIFICATION DOCUMENTS:
*
*   /home/project/3531/Docu/GCB.spec.doc
*
*
* FILE MODULES:
*
*   build_def_fn_input_popup()   - builds the "Select a User Defined Function" popup.
*   build_local_input()          - builds popup to declare or select variables.
*   build_str_popup()            - builds the popup to receive string variable input.
*   build_unknown_type_popup()   - builds popup to receive types of undeclared vars.
*   build_var_input_popup()      - builds the popup to receive number value input.
*   cbr_mat_tgls()               - processes the scalar/matrix/vector toggles.
*   cbr_def_fn()                 - Puts user-defined function name in expression string.
*   cbr_expr_input()             - called when user selects a math/logic button.
*   cbr_expr_num()               - set text string to user's number choice
*   do_parse()                   - set globals, parse expression, reset input buttons
*   init_inputs()                - initializes input buttons based on expression type
*   invalidate_buttons()         - sets buttons (in)active based on parser state.
*   set_state()                  - gets parse state from yacc, sets expression buttons
*   setup_logic_type_popup()     - sets toggles and text fields for variable input
*   setup_math_menu()            - builds the logical expression window
*
*****************************************<----->*****************************************/

#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xm/Xm.h>
#include <Xm/MwmUtil.h>
#include <Xm/SelectioB.h>
#include <Xm/Form.h>

#include "gcb.h"
#include "widgets.h"
#include "menu.h"
#include "symbol.h"
#include "gcb_parse.h"
#include "next_inputs.h"
#include "term_set.h"
#include "constants.h"
#include "expr_menu.h"
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  build_def_fn_input_popup()
*
* MODULE FUNCTION:
*
*   This routine builds the "Select a User Defined Function" popup.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  build_def_fn_input_popup( parent )

    Widget parent;

{
    int         n;
    Arg         args[6];
    XmString    cstr1,
                cstr2,
                cstr3;

    dlg_def_fn = cr_popup( NULLS, parent, "Defined Function Input" );

    cstr1 = XmStringCreate( "Functions", XmSTRING_DEFAULT_CHARSET );
    cstr2 = XmStringCreate( "Cancel",    XmSTRING_DEFAULT_CHARSET );
    cstr3 = XmStringCreate( "Done",      XmSTRING_DEFAULT_CHARSET );

    /*
     *  Build the selection box from which the user will select the "user defined
     *  function."  The "OK" button is usually the left-most button in a selection
     *  box, but we want it to be in the middle because the "CANCEL" button is
     *  always the left-most button in the GCB.
     */

    n = 0;
    XtSetArg( args[n], XmNlistLabelString,  cstr1); n++;
    XtSetArg( args[n], XmNokLabelString,    cstr2); n++;
    XtSetArg( args[n], XmNcancelLabelString,cstr3); n++;
    XtSetArg( args[n], XmNokCallback,       def_fn_cancel_code);n++;
    XtSetArg( args[n], XmNcancelCallback,   def_fn_done_code);  n++;
    XtSetArg( args[n], XmNhelpCallback,     def_fn_help_code);  n++;
    sel_bx_def_fn = XmCreateSelectionBox( dlg_def_fn, NULLS, args, n );
    XtManageChild( sel_bx_def_fn );

    txt_def_fn = XmSelectionBoxGetChild( sel_bx_def_fn, XmDIALOG_TEXT );

    XmStringFree( cstr1 );
    XmStringFree( cstr2 );
    XmStringFree( cstr3 );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  build_local_input()
*
* MODULE FUNCTION:
*
*   This routine builds the popup to declare new variables or select existing ones.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  build_local_input( parent )

    Widget parent;

{
    Arg         args[10];
    XmString    tcs;
    Widget      list;
    int         n = 0;

    dlg_logic_type = cr_popup( NULLS, parent, "Variable Selector" );
    FormW = cr_form( NULLS, dlg_logic_type, NULL, NULL );
    set_attribs( FORM, FormW, 550, 350, XmRESIZE_NONE );

    tcs = XmStringLtoRCreate( "Variables",   XmSTRING_DEFAULT_CHARSET );

    XtSetArg( args[n], XmNlistLabelString,  tcs);                    n++;
    XtSetArg( args[n], XmNtextColumns,      40);                     n++;
    XtSetArg( args[n], XmNokCallback,       var_input_done_code);    n++;
    XtSetArg( args[n], XmNcancelCallback,   var_input_cancel_code);  n++;
    XtSetArg( args[n], XmNhelpCallback,     def_fn_help_code);       n++;
    sel_bx_var_type = XmCreateSelectionBox( FormW, NULLS, args, n );
    XtManageChild( sel_bx_var_type );

    XmStringFree( tcs );

    /*
     *  Replace the automatic list callback with cbr_var_choose so we can
     *  prune the var name from list line before installing it in the
     *  text field.
     */

    txt_var_name = XmSelectionBoxGetChild( sel_bx_var_type, XmDIALOG_TEXT );

    list = XmSelectionBoxGetChild( sel_bx_var_type, XmDIALOG_LIST );
    XtSetArg( args[0], XmNbrowseSelectionCallback,  var_selc_code );
    XtSetValues( list, args, 1 );

    XtUnmanageChild( XmSelectionBoxGetChild(sel_bx_var_type, XmDIALOG_DEFAULT_BUTTON) );
    XtUnmanageChild( XmSelectionBoxGetChild(sel_bx_var_type, XmDIALOG_CANCEL_BUTTON) );
    XtUnmanageChild( XmSelectionBoxGetChild(sel_bx_var_type, XmDIALOG_HELP_BUTTON) );
    XtUnmanageChild( XmSelectionBoxGetChild(sel_bx_var_type, XmDIALOG_SEPARATOR) );

    cr_label( NULLS, FormW, "Data Type", 0, 7, IGNORE, 73, IGNORE);
    cr_separator( NULLS, FormW, 13, IGNORE, 60, 99 );

    XtSetArg( args[0], XmNorientation,      XmHORIZONTAL );
    XtSetArg( args[1], XmNpacking,          XmPACK_COLUMN );
    XtSetArg( args[2], XmNnumColumns,       2 );
    XtSetArg( args[3], XmNradioBehavior,    True );
    rb_var_type= (Widget) XmCreateRadioBox( FormW, NULLS, args, 4 );
    XtManageChild( rb_var_type );

    tgl_var_type_int      = cr_toggle( NULLS, rb_var_type, "Int",       NULL, 1, 1 );
    tgl_var_type_real     = cr_toggle( NULLS, rb_var_type, "float",     NULL, 1, 1 );
    tgl_var_type_string   = cr_toggle( NULLS, rb_var_type, "String",    NULL, 1, 1 );
    tgl_var_type_unsigned = cr_toggle( NULLS, rb_var_type, "Unsigned",  NULL, 1, 1 );
    tgl_var_type_short    = cr_toggle( NULLS, rb_var_type, "Short",     NULL, 1, 1 );
    tgl_var_type_double   = cr_toggle( NULLS, rb_var_type, "Double",    NULL, 1, 1 );

    XtAddCallback( tgl_var_type_int,      XmNarmCallback, cbr_var_type, INTEGER);
    XtAddCallback( tgl_var_type_real,     XmNarmCallback, cbr_var_type, FLOAT);
    XtAddCallback( tgl_var_type_string,   XmNarmCallback, cbr_var_type, CHAR);
    XtAddCallback( tgl_var_type_unsigned, XmNarmCallback, cbr_var_type, UNSIGNED);
    XtAddCallback( tgl_var_type_short,    XmNarmCallback, cbr_var_type, SHORT);
    XtAddCallback( tgl_var_type_double,   XmNarmCallback, cbr_var_type, DOUBLE);

    cr_label    ( NULLS, FormW, "Scalar/Matrix/Vector", 0, 38, IGNORE, 68, IGNORE );
    cr_separator( NULLS, FormW, 44, IGNORE, 60, 99 );

    rb_isit_mat  = cr_radio_box( NULLS, FormW, XmHORIZONTAL );
    tgl_isnt_mat = Ncr_toggle( NULLS,rb_isit_mat, "Scalar",        cbr_mat_tgls, SCAL,0);
    tgl_is_mat   = Ncr_toggle( NULLS,rb_isit_mat, "Matrix/Vector", cbr_mat_tgls, MAT, 0);

    txt_matdim_x = cr_text( NULLS, FormW, NULL, NULL, NULLS, FALSE, 1, 2 );
    txt_matdim_y = cr_text( NULLS, FormW, NULL, NULL, NULLS, FALSE, 1, 2 );
    txt_matdim_3 = cr_text( NULLS, FormW, NULL, NULL, NULLS, FALSE, 1, 2 );
    txt_matdim_4 = cr_text( NULLS, FormW, NULL, NULL, NULLS, FALSE, 1, 2 );
    txt_matelm_x = cr_text( NULLS, FormW, NULL, NULL, NULLS, FALSE, 1, 2 );
    txt_matelm_y = cr_text( NULLS, FormW, NULL, NULL, NULLS, FALSE, 1, 2 );
    txt_matelm_3 = cr_text( NULLS, FormW, NULL, NULL, NULLS, FALSE, 1, 2 );
    txt_matelm_4 = cr_text( NULLS, FormW, NULL, NULL, NULLS, FALSE, 1, 2 );

    cr_label( NULLS, FormW, "Matrix Size",      0, 56, IGNORE, 60, IGNORE );
    cr_label( NULLS, FormW, "Matrix Element",   0, 66, IGNORE, 60, IGNORE );

    cr_separator( NULLS, FormW, 80, IGNORE, 1, 99 );

    DoneW   = cr_command( NULLS, FormW, "Done",   cbr_var_input_done,DONE );
    CancelW = cr_command( NULLS, FormW, "Cancel",cbr_var_input_done,CANCEL );
    HelpW   = cr_command( NULLS, FormW, "Help",   cbr_help,          VARIABLE_HELP );

    set_position( rb_var_type,  17, IGNORE, 60, IGNORE);
    set_position( rb_isit_mat,  45, IGNORE, 60, IGNORE);
    set_position( txt_matdim_x, 55, IGNORE, 78, IGNORE);
    set_position( txt_matdim_y, 55, IGNORE, 83, IGNORE);
    set_position( txt_matdim_3, 55, IGNORE, 88, IGNORE);
    set_position( txt_matdim_4, 55, IGNORE, 93, IGNORE);
    set_position( txt_matelm_x, 65, IGNORE, 78, IGNORE);
    set_position( txt_matelm_y, 65, IGNORE, 83, IGNORE);
    set_position( txt_matelm_3, 65, IGNORE, 88, IGNORE);
    set_position( txt_matelm_4, 65, IGNORE, 93, IGNORE);
    set_position( CancelW,      89, IGNORE,  5, IGNORE);
    set_position( DoneW,        89, IGNORE, 40, IGNORE);
    set_position( HelpW,        89, IGNORE, 75, IGNORE);
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  build_str_popup()
*
* MODULE FUNCTION:
*
*   This routine  builds the popup to receive string variable input.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  build_str_popup( parent )

    Widget parent;

{
    dlg_str_input = cr_popup( NULLS, parent, "String Input" );

    FormW = cr_form( NULLS, dlg_str_input, NULL, NULL );
    set_attribs( FORM, FormW, 500, 125, XmRESIZE_NONE );

    cr_label( NULLS, FormW, "Enter a string", 0, 3, 20, 5, 95 );

    txt_str_input = cr_text( NULLS, FormW, NULL, NULL, NULLS, FALSE, 1, 60 );

    cr_separator( NULLS, FormW, 70, IGNORE, 1, 99 );

    CancelW = cr_command( NULLS, FormW, "Cancel",cbr_str_input_done, CANCEL );
    DoneW   = cr_command( NULLS, FormW, "Done",  cbr_str_input_done, DONE   );
    HelpW   = cr_command( NULLS, FormW, "Help",  cbr_help,           STRING_HELP );

    set_position( txt_str_input,    30, IGNORE, 15, IGNORE);
    set_position( CancelW,          80, IGNORE,  5, IGNORE);
    set_position( DoneW,            80, IGNORE, 40, IGNORE);
    set_position( HelpW,            80, IGNORE, 75, IGNORE);
}


/*****************************************<----->*****************************************
*
* MODULE NAME:  build_unknown_type_popup()
*
* MODULE FUNCTION:
*
*   This routine builds the popup to receive the types of undeclared variables.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  build_unknown_type_popup(parent)

    Widget parent;

{
    Arg     args[5];

    dlg_unknown_type = cr_popup( NULLS, parent, "Data Type Declaration" );

    FormW = cr_form( NULLS, dlg_unknown_type, NULL, NULL );
    set_attribs( FORM, FormW, 650, 350, XmRESIZE_NONE );

    cr_label( NULLS, FormW, "Undeclared Variables", 0,  5, IGNORE,  5, IGNORE );
    cr_label( NULLS, FormW, "Data Types",           0, 10, IGNORE, 72, IGNORE );
    cr_separator( NULLS, FormW, 12, IGNORE,  5, 27 );
    cr_separator( NULLS, FormW, 15, IGNORE, 60, 97 );

    rb_unknown_type_name  = cr_radio_box( NULLS, FormW, XmVERTICAL );
    set_position( rb_unknown_type_name, 17, IGNORE, 10, IGNORE);

    tgl_unknown_type_name[0] = Ncr_toggle ( NULLS, rb_unknown_type_name, NULLS,
        cbr_unknown_type_select_name, 0, 0);
    tgl_unknown_type_name[1] = Ncr_toggle ( NULLS, rb_unknown_type_name, NULLS,
        cbr_unknown_type_select_name, 1, 1);
    tgl_unknown_type_name[2] = Ncr_toggle ( NULLS, rb_unknown_type_name, NULLS,
        cbr_unknown_type_select_name, 2, 2);
    tgl_unknown_type_name[3] = Ncr_toggle ( NULLS, rb_unknown_type_name, NULLS,
        cbr_unknown_type_select_name, 3, 3);
    tgl_unknown_type_name[4] = Ncr_toggle ( NULLS, rb_unknown_type_name, NULLS,
        cbr_unknown_type_select_name, 4, 4);

    lbl_unknown_type_name[0] = cr_label( NULLS, FormW, NULLS, 0, 20, IGNORE, 28, IGNORE);
    lbl_unknown_type_name[1] = cr_label( NULLS, FormW, NULLS, 0, 30, IGNORE, 28, IGNORE);
    lbl_unknown_type_name[2] = cr_label( NULLS, FormW, NULLS, 0, 41, IGNORE, 28, IGNORE);
    lbl_unknown_type_name[3] = cr_label( NULLS, FormW, NULLS, 0, 51, IGNORE, 28, IGNORE);
    lbl_unknown_type_name[4] = cr_label( NULLS, FormW, NULLS, 0, 62, IGNORE, 28, IGNORE);

    XtSetArg( args[0], XmNorientation,  XmHORIZONTAL  );
    XtSetArg( args[1], XmNpacking,      XmPACK_COLUMN );
    XtSetArg( args[2], XmNnumColumns,   2 );
    XtSetArg( args[3], XmNradioBehavior, True );
    rb_unknown_type_type= (Widget)XmCreateRadioBox(FormW, NULLS, args, 4);
    XtManageChild(rb_unknown_type_type);
    set_position( rb_unknown_type_type, 17, IGNORE, 60, IGNORE);

    tgl_unknown_type_int =    Ncr_toggle( NULLS, rb_unknown_type_type, "Int",
        cbr_unknown_type_select_type, 0, 0);
    tgl_unknown_type_float =  Ncr_toggle(NULLS, rb_unknown_type_type, "Float",
        cbr_unknown_type_select_type, 1, 1);
    tgl_unknown_type_double = Ncr_toggle(NULLS,rb_unknown_type_type, "Double",
        cbr_unknown_type_select_type, 2, 2);
    tgl_unknown_type_short = Ncr_toggle(NULLS,rb_unknown_type_type, "Short",
        cbr_unknown_type_select_type, 3, 3);
    tgl_unknown_type_unsigned = Ncr_toggle(NULLS,rb_unknown_type_type,"Unsigned",
        cbr_unknown_type_select_type, 4, 4);
    tgl_unknown_type_string = Ncr_toggle(NULLS,rb_unknown_type_type,"String",
        cbr_unknown_type_select_type, 5, 5);

    cr_label ( NULLS, FormW, "Matrix?", 0, 42, IGNORE, 62, IGNORE);
    rb_unknown_isit_mat  = cr_radio_box( NULLS, FormW, XmHORIZONTAL );
    set_position( rb_unknown_isit_mat,  40, IGNORE, 75, IGNORE);
    tgl_unknown_is_mat   = Ncr_toggle  ( NULLS, rb_unknown_isit_mat, "Yes",
        cbr_unknown_is_mat, 0, 0);
    tgl_unknown_isnt_mat = Ncr_toggle  ( NULLS, rb_unknown_isit_mat, "No",
        cbr_unknown_is_mat, 1, 1);

    txt_mat_numrows =   cr_text( NULLS, FormW, NULL, NULL, NULLS, FALSE, 1, 2);
    txt_mat_numcols =   cr_text( NULLS, FormW, NULL, NULL, NULLS, FALSE, 1, 2);
```

```
    txt_mat_dim3    =   cr_text( NULLS, FormW, NULL, NULL, NULLS, FALSE, 1, 2);
    txt_mat_dim4    =   cr_text( NULLS, FormW, NULL, NULL, NULLS, FALSE, 1, 2);
    set_position( txt_mat_numrows,  55, IGNORE, 68, IGNORE);
    set_position( txt_mat_numcols,  55, IGNORE, 81, IGNORE);
    set_position( txt_mat_dim3, 65, IGNORE, 68, IGNORE);
    set_position( txt_mat_dim4, 65, IGNORE, 81, IGNORE);

    cr_label( NULLS, FormW, "Dim1",     0, 56, IGNORE, 62, IGNORE);
    cr_label( NULLS, FormW, "Dim2",     0, 56, IGNORE, 75, IGNORE);
    cr_label( NULLS, FormW, "Dim3",     0, 66, IGNORE, 62, IGNORE);
    cr_label( NULLS, FormW, "Dim4",     0, 66, IGNORE, 75, IGNORE);

    cr_separator( NULLS, FormW, 82, IGNORE, 5, 95 );

    DoneW = cr_command( NULLS, FormW, "Done", cbr_unknown_type_done, 1 );
    HelpW = cr_command( NULLS, FormW, "Help", cbr_help,              CREATE_ELEM);
    CancelW = cr_command( NULLS, FormW, "Cancel", cbr_unknown_type_done,  0 );

    set_position( CancelW, 89, IGNORE, 10, IGNORE);
    set_position( DoneW, 89, IGNORE, 40, IGNORE);
    set_position( HelpW, 89, IGNORE, 70, IGNORE);
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  build_var_input_popup()
*
* MODULE FUNCTION:
*
*   This routine builds the popup to receive number value input.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                                  Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  build_var_input_popup(parent)

    Widget parent;

{
    int         n;
    Arg         args[10];
    char        *var_string,  *num_string, *str_string;

    dlg_var_input = cr_popup( NULLS, parent, "Variable Input" );

    FormW = cr_form( NULLS, dlg_var_input, NULL, NULL );
    set_attribs( FORM, FormW, 300, 325, XmRESIZE_NONE );

    num_string = (char *)malloc( sizeof(char)*(34) );
    strcat( num_string, "Please enter a valid number below.\n");

    str_string = (char *)malloc( sizeof(char)*(34) );
    strcat( str_string, "Please enter a valid string below.\n");

    num_header = XmStringLtoRCreate(  (char *)num_string, XmSTRING_DEFAULT_CHARSET);
    str_header = XmStringLtoRCreate(  (char *)str_string, XmSTRING_DEFAULT_CHARSET);

    free(num_string);
    free(str_string);

    lbl_var_input_header = cr_label(NULLS, FormW, NULLS, 0, 3, 20, 5, 95);

    scr_var_input = cr_text(NULLS, FormW, NULL, NULL, NULLS, FALSE, 1, 30);
    set_position(scr_var_input, 20, IGNORE, 20, IGNORE);

    rc_expr_num = cr_rowcol( NULLS, FormW, 4, XmHORIZONTAL, NULL, NULL);
    set_position( rc_expr_num, 35, IGNORE, 35, IGNORE );

    BtnW = cr_command(NULLS, rc_expr_num, " 7 ", cbr_expr_num, "7");
    BtnW = cr_command(NULLS, rc_expr_num, " 8 ", cbr_expr_num, "8");
    BtnW = cr_command(NULLS, rc_expr_num, " 9 ", cbr_expr_num, "9");
    BtnW = cr_command(NULLS, rc_expr_num, " 4 ", cbr_expr_num, "4");
    BtnW = cr_command(NULLS, rc_expr_num, " 5 ", cbr_expr_num, "5");
    BtnW = cr_command(NULLS, rc_expr_num, " 6 ", cbr_expr_num, "6");
    BtnW = cr_command(NULLS, rc_expr_num, " 1 ", cbr_expr_num, "1");
    BtnW = cr_command(NULLS, rc_expr_num, " 2 ", cbr_expr_num, "2");
    BtnW = cr_command(NULLS, rc_expr_num, " 3 ", cbr_expr_num, "3");
    BtnW = cr_command(NULLS, rc_expr_num, " 0 ", cbr_expr_num, "0");
    BtnW = cr_command(NULLS, rc_expr_num, " . ", cbr_expr_num, ".");

    cr_separator( NULLS, FormW, 80, IGNORE, 1, 99 );
```

```
    DoneW   = cr_command( NULLS,FormW,"Done",   cbr_num_input_done,CBR_COMP_TYPE_DONE);
    CancelW = cr_command( NULLS,FormW,"Cancel",cbr_num_input_done,CBR_COMP_TYPE_CANCEL);

    HelpW = cr_command( NULLS, FormW, "Help", cbr_help, NUMBER_HELP );

    set_position( CancelW,  89, IGNORE,  3, IGNORE );
    set_position( DoneW,    89, IGNORE, 37, IGNORE );
    set_position( HelpW,    89, IGNORE, 70, IGNORE );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_mat_tgls()
*
* MODULE FUNCTION:
*
*   This routine processes the user's selection of the Scalar/Matrix/Vector toggle
*   buttons.  This routine will determine which type of variable the user wants
*   to select, and will then control the matrix size/element text fields accordingly.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_mat_tgls( w, client_data, call_data )

    Widget      w;
    int         client_data;
    caddr_t     call_data;

{
    int     sym_type;
    Arg     args[1];

    XtSetArg( args[0], XmNuserData, &sym_type );
    XtGetValues( dlg_logic_type, args, 1 );

    /*
     *  User clicked Scalar.
     */

    if ( client_data == SCAL )
        {
        /*
         *  Clear the matrix size/elements text widgets, they are not appropriate
         *  at this point.  Set the matrix size/element text widgets inactive.
         */

        XmTextSetString( txt_matdim_x, NULL );
        XmTextSetString( txt_matdim_y, NULL );
        XmTextSetString( txt_matdim_3, NULL );
        XmTextSetString( txt_matdim_4, NULL );

        XmTextSetString( txt_matelm_x, NULL );
        XmTextSetString( txt_matelm_y, NULL );
        XmTextSetString( txt_matelm_3, NULL );
        XmTextSetString( txt_matelm_4, NULL );

        XmTextSetEditable( txt_matdim_x, False );
        XmTextSetEditable( txt_matdim_y, False );
        XmTextSetEditable( txt_matdim_3, False );
        XmTextSetEditable( txt_matdim_4, False );

        XmTextSetEditable( txt_matelm_x, False );
        XmTextSetEditable( txt_matelm_y, False );
        XmTextSetEditable( txt_matelm_3, False );
        XmTextSetEditable( txt_matelm_4, False );

        XmTextSetEditable( txt_var_name, True );
```

```
        return;
        }

    /*
     *  User clicked matrix.
     */

    if ( client_data == MAT )
        {
        XmTextSetEditable( txt_matdim_x, True );
        XmTextSetEditable( txt_matelm_x, True );

        /*
         *  If the user is selection a local or global variable, then let them
         *  fill in both dimensions.  If they are selection an Object or WS Global,
         *  then only let them select the number of rows, columns must == 1.
         */

        if ((sym_type == LOCAL_VAR) || (sym_type == GLOBAL_VAR))
            {
            XmTextSetEditable( txt_matdim_y, True );
            XmTextSetEditable( txt_matdim_3, True );
            XmTextSetEditable( txt_matdim_4, True );
            XmTextSetEditable( txt_matelm_y, True );
            XmTextSetEditable( txt_matelm_3, True );
            XmTextSetEditable( txt_matelm_4, True );
            }
        else
            {
            XmTextSetString( txt_matdim_y, "1" );
            XmTextSetString( txt_matdim_3, "1" );
            XmTextSetString( txt_matdim_4, "1" );
            XmTextSetString( txt_matelm_y, "1" );
            XmTextSetString( txt_matelm_3, "1" );
            XmTextSetString( txt_matelm_4, "1" );
            }

        XmTextSetEditable( txt_var_name, True );
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_def_fn()
*
* MODULE FUNCTION:
*
*   This routine is called when the user selects a defined function from the
*   selection list.  This routine inserts the function name into the expression
*   string.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_def_fn( w, list, call_data )

    Widget                  w;
    Widget                  list;
    XmListCallbackStruct    *call_data;

{
    char    newStr[ MAX_NAME ],
            *string;
    int     attributes = 0,
            i,
            len;

    /*
     *  Get the function name selected by the user.
     */

    XmStringGetLtoR( call_data->item, XmSTRING_DEFAULT_CHARSET, &string );

    /*
     *  Copy the string to a local string, we are going to modify the string
     *  and we don't want to modify the string Motif just gave us a pointer to.
     *  Set a pointer to the start of the extension, then set it to NULL to
     *  to remove the extension.
     */

    strcpy( newStr, string );
    len = strlen( newStr );

    for ( i=len-1; i>0; i-- )
        if ( newStr[i] == '.' )
            {
            newStr[i] = NULL;
            break;
            }

    /*
     *  See if the user defined function is in the symbol table, if it is not,
     *  add it.
     */

    if (! (struct symbol_entry *) lookup_symbol(NULL,newStr) )
        {

        /*DEBUG*/
```

```
        elog(3,"cbr def: adding fn %s to symtab", newStr);

        attributes |= PROCEDURE;
        attributes |= INTEGER;
        if ( add_symbol_entry(NULL,newStr,attributes,0,0,0,0,0) )
            error_handler( ERR_ADD_SYMBOL, "cbr_def_fn" );
        }

    /*
     *  Insert the string which now contains only the function name (no extension)
     *  into the expression.
     */

    XmTextInsert( scr_expr, XmTextGetInsertionPosition(scr_expr), newStr );
    XtUnmanageChild( dlg_def_fn );
    do_parse();
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_expr_input()
*
* MODULE FUNCTION:
*
*   This routine is called when the user selects one of the buttons from the
*   math/logic menu of tokens.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_expr_input( w, closure, call_data )

    Widget      w;
    int         closure;
    caddr_t     call_data;

{
    Arg         args[1];
    static int  last_closure = LPAR;
    int         sym_type;

    /*
     *  Insert a blank space into the expression everywhere except the first
     *  position and after parens.
     */

    if ( (closure != RPAR) && (last_closure != LPAR) &&
        XmTextGetInsertionPosition(scr_expr) )
            XmTextInsert(scr_expr, XmTextGetInsertionPosition(scr_expr), " ");

    last_closure = closure;

    /*
     *  Find out which button called us, many of the simpler buttons are
     *  handled in the "default" case.
     */

    switch ( closure )
        {
        case STRING :

            /*
             *  put up popup to receive string input.
             */

            XtManageChild( dlg_str_input );
            return;

        case MSID       :
        case LOCAL_VAR  :
        case GLOBAL     :
        case WSG        :

            /*
             *  set userData for the dlg widget so cbr_var_input can retrieve
             *  it and know what button caused it to be popped up.
```

```
        */

        if ( closure == GLOBAL )
            XtSetArg( args[0], XmNuserData, GLOBAL_VAR );
        else if ( closure == MSID )
            XtSetArg( args[0], XmNuserData, WS_OBJECT );
        else if ( closure == WSG )
            XtSetArg( args[0], XmNuserData, WS_GLOBAL );
        else
            XtSetArg( args[0], XmNuserData, closure );
        XtSetValues( dlg_logic_type, args, 1 );
        cbr_mat_tgls( NULL, SCAL, NULL );

        /*
         *  load variable list, set prefix in text string, and (dis)arm
         *  toggles
         */

        setup_logic_type_popup( closure );
        XtManageChild( dlg_logic_type );
        break;

    case NUMBER :

        /*
         *  Set up header in var input popup
         */

        XtSetArg( args[0], XmNlabelString, num_header );
        XtSetValues( lbl_var_input_header, args, 1 );
        XtManageChild( dlg_var_input );
        break;

    case DEF_FN :

        /*
         *  load defined functions into popup list.
         */

        load_fn_list( sel_bx_def_fn );

        /*
         *  Set prefix of user def fn
         */

        XmTextSetString( txt_def_fn, "FN_" );
        XtSetArg( args[0], XmNcursorPosition, (XmTextPosition) 4 );
        XtSetValues( txt_def_fn, args, 1 );

        XtManageChild( dlg_def_fn );
        break;

    case TRIG :
        break;

    default :
        if ( closure == EQ )
            {

            /*
             *  insert a := if we are in a set, = otherwise.
             */

            XtSetArg( args[0], XmNuserData, &sym_type );
            XtGetValues( current_symbol, args, 1 );
            if ( sym_type == SET )
                XmTextInsert( scr_expr, XmTextGetInsertionPosition(scr_expr), ":=" );
            else
                XmTextInsert( scr_expr, XmTextGetInsertionPosition(scr_expr), "=" );
            }
        else

            /*
             *  insert string from global b_strings array.
             */

            XmTextInsert( scr_expr, XmTextGetInsertionPosition(scr_expr),
                b_strings[closure-5] );

        /*
         *  parse expression so far; reset expression buttons.
         */

        do_parse();
        break;
    }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_expr_num()
*
* MODULE FUNCTION:
*
*   This routine sets text string to user's number choice in number input popup
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_expr_num( w, client_data, call_data )

    Widget      w;
    caddr_t     client_data;
    caddr_t     call_data;

{
    XmTextSetString( scr_var_input,
        strcat(XmTextGetString(scr_var_input), (char *)client_data) );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  do_parse()
*
* MODULE FUNCTION:
*
*   This routine sets necessary globals, parses expression, resets input buttons
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  do_parse()

{
    int     sym_type;
    Arg     args[1];
    int     i;

    XtSetArg( args[0], XmNuserData, &sym_type );
    XtGetValues( current_symbol, args, 1 );

    /*
     *  Set global so parser will know what kind of expression we are parsing.
     */

     if ( sym_type == SET )
        SetSym = 1;
     else if ( sym_type == IF )
        SetSym = 0;
    else
        {
        /*DEBUG*/
        elog(1,"do_parse: type of current symbol is neither if nor set");
        exit( ERR );
        }

     i = parse_expression( XmTextGetString(scr_expr) );
     if ( (i != PARSE_SUCCESS) && (i != END_OF_FILE) )
        {
        elog(3,"in do_parse, putting up user ack after parse returned %i", i);
        user_ack("Syntax error in expression");

        /*
         *  force user to correct error by hand by making all expression
         *  buttons inactive.
         */

        init_inputs( -3 );
        return( 0 );
        }

    /*
     *  reset active buttons based on parser state.
     */

    invalidate_buttons();
    return( 1 );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  init_inputs()
*
* MODULE FUNCTION:
*
*   This routine sets the initial state of the panel depending on the palette
*   item selected.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  init_inputs( type )

    int type;

{
    int i;

    for ( i=0; i<NUM_BUTTONS; i++ )
            active_list[i] = 0;

    if ( type == -1 )
        {

        /*
         *  initial state - if
         */

        active_list[NOT] = 1;
        active_list[LPAR] = 1;
        active_list[NUMBER] = 1;
        active_list[LOCAL] = 1;
        active_list[MSID] = 1;
        active_list[GLOBAL] = 1;
        active_list[PLUS] = 1;
        active_list[MINUS] = 1;
        active_list[PI] = 1;
        active_list[SQRT] = 1;
        active_list[LOG] = 1;
        active_list[NLOG] = 1;
        active_list[TRIG] = 1;
        active_list[POWER] = 1;
        active_list[EXP] = 1;
        active_list[MINUS] = 1;
        active_list[DEF_FN] = 1;
        active_list[WSG] = 1;
        }
    else if ( type == -2 )

        /*
         *  initial state - set
         */

        {
        active_list[LOCAL] = 1;
        active_list[GLOBAL] = 1;
        active_list[WSG] = 1;
```

```
        }

    else if ( type == -3 )
        {

        /*
         *  parse of existing expression failed; turn off all buttons
         *  (don't set any bits of the active list).
         */

        }
    else
        {
        user_ack("Fatal error: init_inputs() bad button type");
        elog(1,"init_inputs: received a bad button type");
        exit( ERR );
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  invalidate_buttons()
*
* MODULE FUNCTION:
*
*   This routine renders inactive all but the buttons indicated in the
*   valid_points array.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void invalidate_buttons()

{
    int     i;
    Arg     active_args[2], inactive_args[2];
    extern int  insensitive_color;

    /*
     *  2 sets of args so we don't have to reset them inside loop.
     */

    XtSetArg( active_args[0], XmNsensitive,  (XtArgVal) TRUE );
    XtSetArg( active_args[1], XmNbackground,
        (XtArgVal) WhitePixel(display, DefaultScreen(display)) );

    XtSetArg( inactive_args[0], XmNsensitive,  (XtArgVal) FALSE );
    XtSetArg( inactive_args[1], XmNbackground, (XtArgVal) insensitive_color );

    for (i=0; i<NUM_BUTTONS; i++)
        {
        if (!active_list[i])
             XtSetValues( w[i], inactive_args, 2 );
        else XtSetValues( w[i], active_args, 2 );
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  set_state()
*
* MODULE FUNCTION:
*
*   This routine is called from yacc.src; it sets the expression buttons
*   (in)active based on the parser state.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  set_state( state, terminal )

    int state, terminal;

{
    int i;

elog( 2, "set state: state is %i\n", state );

    /*
     * special case for id on left side of set stmt
     */

    if ( (WhereAmI == LHS) && (SetSym) )
        {
        for ( i=0; i<NUM_BUTTONS; i++ )
            active_list[i] = 0;
        active_list[EQ] = 1;
        return;
        }

    for ( i=0; i<NUM_BUTTONS; i++ )
        {
        if ( i == RPAR )
            {

            /*
             *  set rpar active only if parens are unbalanced
             */

            /*DEBUG*/
            elog(3,"paren_count = %d", paren_count);

            if ( !paren_count )
                active_list[i] = 0;
            else
                {
                if ( term_set[state][i] )
                    active_list[i] = 1;
                else if ( terminal )
                    active_list[i] = 0;
                }
            }

        else if ( i == STRING )
            {

            /*
             *  Don't allow String types in IF expressions.
             */

            if ( !SetSym )
                active_list[i] = 0;
            else
                {

                /*
                 *  if we should turn the STRING button on, turn it on,
                 *  else turn it off.
                 */

                if ( term_set[state][i] )
                    active_list[i] = 1;
                else
                    {
                    if ( terminal )
                        active_list[i] = 0;
                    }
                }
            }
        else if ( (i >= EQ) && (i <= GT) )
            {

            /*
             *  set relops active only if on left side and parens are balanced
             */

            if ( (paren_count) || (WhereAmI == RHS) )
                active_list[i] = 0;
            else
                {
                if ( term_set[state][i] )
                    active_list[i] = 1;
                else if ( terminal )
                    active_list[i] = 0;
                }
            }
        else if ( (i == AND) || (i == OR) )
            {

            /*
             *  set logops active only if on right side and parens are balanced
             */

            if ( (paren_count) || (WhereAmI == LHS) )
                active_list[i] = 0;
            else if ( SetSym )
                active_list[i] = 0;
            else
                {
                if ( term_set[state][i] )
                    active_list[i] = 1;
                else if ( terminal )
                    active_list[i] = 0;
                }
            }
        else
            {

            /*
```

```
            *  if the button should be active according to the current state,
            *  set the corresponding bit in active_list
            */

           if ( term_set[state][i] )
               active_list[i] = 1;
           else
               {
               if ( terminal )

               /*
                *  we have reached a terminal in the parse; reset active list
                *  for non-terminals, the state is the aggregate of all states
                *  leading to it.
                */
                  active_list[i] = 0;
               }
           }
       }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  setup_math_menu()
*
* MODULE FUNCTION:
*
*   This routine builds the logical expression window that overlays the
*   palette items.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  setup_math_menu(parent, under)

    Widget parent, under;

{
    int     n, m = 0;
    Arg     args[10];

    /*
     *  create Frame to hold math rowcol widget
     */

    frame_math_menu = cr_frame( NULLS, parent, NULL, under);
    XtSetArg( args[0], XtNmappedWhenManaged, FALSE );
    XtSetValues( frame_math_menu, args, 1 );

    /*
     *  create form to hold rowcols and labels
     */

    n = 0;
    XtSetArg( args[n], XmNwidth,  278 ); n++;
    XtSetArg( args[n], XmNheight, 464 ); n++;
    frm_math_menu = XmCreateForm( frame_math_menu, NULLS, args, n);
    XtManageChild( frm_math_menu );

    /*
     *  create labels in math menu
     */

    cr_label( NULLS, frm_math_menu, "GCB Elements", 1,  1,  7, 1, 97 );
    cr_label( NULLS, frm_math_menu, "Logic",         0,  9, 12, 1, 99 );
    cr_label( NULLS, frm_math_menu, "Variable",      0, 28, 31, 1, 99 );
    cr_label( NULLS, frm_math_menu, "Relational",    0, 48, 52, 1, 99);
    cr_label( NULLS, frm_math_menu, "Math",          0, 62, 65, 1, 99 );


    /*
     *  create rowcol to hold palette variable items
     */

    cr_separator( NULLS, frm_math_menu, 32, 33, 1, 97 );
    rc_var  = cr_rowcol( NULLS, frm_math_menu, 2, XmHORIZONTAL, NULL, NULL);
    set_position(rc_var, 33, IGNORE, 12, IGNORE);

    w[m] = cr_command(NULLS, rc_var, "Object", cbr_expr_input, MSID); m++;
```

```
w[m] = cr_command(NULLS, rc_var, "global", cbr_expr_input, GLOBAL);    m++;
w[m] = cr_command(NULLS, rc_var, "local",  cbr_expr_input, LOCAL_VAR); m++;
w[m] = cr_command(NULLS, rc_var, "number", cbr_expr_input, NUMBER);    m++;
w[m] = cr_command(NULLS, rc_var, "string", cbr_expr_input, STRING);    m++;

/*
 *  create rowcol to hold palette logic items
 */

cr_separator( NULLS, frm_math_menu, 13, 14, 1, 97 );
rc_logic = cr_rowcol( NULLS, frm_math_menu, 2, XmHORIZONTAL, NULL, NULL);
set_position(rc_logic, 15, IGNORE, 20, IGNORE);

/*
 *  create push buttons in logic menu
 */

w[m] = cr_command(NULLS, rc_logic, "and", cbr_expr_input, m);   m++;
w[m] = cr_command(NULLS, rc_logic, "or", cbr_expr_input, m);  m++;
w[m] = cr_command(NULLS, rc_logic, "not", cbr_expr_input, m);  m++;
w[m] = cr_command(NULLS, rc_logic, "bitAnd", cbr_expr_input, m);  m++;
w[m] = cr_command(NULLS, rc_logic, "bitOr", cbr_expr_input, m);  m++;
w[m] = cr_command(NULLS, rc_logic, "bitXor", cbr_expr_input, m);  m++;

/*
 *  create rowcol to hold palette relational items
 */

cr_separator( NULLS, frm_math_menu, 52, 53, 1, 97 );
rc_rel = cr_rowcol( NULLS, frm_math_menu, 1, XmHORIZONTAL, NULL, NULL);
set_position(rc_rel, 53, IGNORE, 20, IGNORE);

eq_btn = w[m] = cr_command(NULLS, rc_rel, "=", cbr_expr_input, m);   m++;
w[m] = cr_command(NULLS, rc_rel, "<>", cbr_expr_input, m);  m++;
w[m] = cr_command(NULLS, rc_rel, "<=", cbr_expr_input, m);  m++;
w[m] = cr_command(NULLS, rc_rel, ">=", cbr_expr_input, m);  m++;
w[m] = cr_command(NULLS, rc_rel, "<", cbr_expr_input, m);  m++;
w[m] = cr_command(NULLS, rc_rel, ">", cbr_expr_input, m);  m++;

/*
 *  create rowcol to hold palette math items
 */

cr_separator( NULLS, frm_math_menu, 66, 67, 1, 97 );
rc_math = cr_rowcol( NULLS, frm_math_menu, 3, XmHORIZONTAL, NULL, NULL);
set_position(rc_math, 67, IGNORE, 5, IGNORE);

w[m] = cr_command(NULLS, rc_math, "+", cbr_expr_input, m);  m++;
w[m] = cr_command(NULLS, rc_math, "-", cbr_expr_input, m);  m++;
w[m] = cr_command(NULLS, rc_math, "*", cbr_expr_input, m);  m++;
w[m] = cr_command(NULLS, rc_math, "/", cbr_expr_input, m);  m++;
w[m] = cr_command(NULLS, rc_math, "(", cbr_expr_input, m);  m++;

w[m] = cr_command(NULLS, rc_math, ")", cbr_expr_input, m);  m++;
w[m] = cr_command(NULLS, rc_math, ",", cbr_expr_input, m);  m++;
w[m] = cr_command(NULLS, rc_math, "shiftL", cbr_expr_input, m);  m++;
w[m] = cr_command(NULLS, rc_math, "shiftR", cbr_expr_input, m);  m++;
w[m] = cr_command(NULLS, rc_math, "PI", cbr_expr_input, m);  m++;

w[m] = cr_command(NULLS, rc_math, "exp",   cbr_expr_input, m);  m++;
w[m] = cr_command(NULLS, rc_math, "power", cbr_expr_input, m);  m++;
w[m] = cr_command(NULLS, rc_math, "sqrt",  cbr_expr_input, m);  m++;
w[m] = cr_command(NULLS, rc_math, "log",   cbr_expr_input, m);  m++;
w[m] = cr_command(NULLS, rc_math, "nlog",  cbr_expr_input, m);  m++;

/*
 *  create rowcol to hold defined fn item
 */

rc_def_fn = cr_rowcol(NULLS, frm_math_menu, 2, XmHORIZONTAL, NULL, NULL);
set_position( rc_def_fn, 87, IGNORE, 15, IGNORE );

w[m] = cr_command(NULLS, rc_def_fn, "trigonometric",  cbr_trig,       T_SHOW); m++;
w[m] = cr_command(NULLS, rc_def_fn, "quaternion",     cbr_quaternion, T_SHOW); m++;
w[m] = cr_command(NULLS, rc_def_fn, "user defined fn",cbr_expr_input, DEF_FN); m++;
w[m] = cr_command(NULLS, rc_def_fn, "matrix function",cbr_matrix,     M_SHOW); m++;

w[m] = cr_command(NULLS, rc_var,    "WS Global", cbr_expr_input, WSG); m++;
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  setup_var_popup()
*
* MODULE FUNCTION:
*
*   This routine sets up the popup to receive a variable declaration.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  setup_logic_type_popup( type )

    int type;

{
    Arg args[1];

    /*
     *  load the variable list within the popup based on the type of
     *  variable being selected.
     */

    load_variable_list( sel_bx_var_type, type );

    /*
     *  depending on the type of the popup, set the prefix of the
     *  variable.
     */

    if ( type == GLOBAL )
        {
        XmTextSetString( txt_var_name, "GV_" );
        XtSetArg( args[0], XmNcursorPosition, (XmTextPosition) 4 );
        XtSetValues( txt_var_name, args, 1);
        }
    else if ( type == WSG )
        {
        XmTextSetString( txt_var_name, "WS_" );
        XtSetArg( args[0], XmNcursorPosition, (XmTextPosition) 4 );
        XtSetValues( txt_var_name, args, 1);
        }
    else if ( type == MSID )
        {
        XmTextSetString( txt_var_name, "V" );
        XtSetArg( args[0], XmNcursorPosition, (XmTextPosition) 2 );
        XtSetValues( txt_var_name, args, 1);
        }

    /*
     *  The default is not a matrix - set the toggles.
     */

       arm_tgl(tgl_isnt_mat);
    disarm_tgl(tgl_is_mat);

    /*
     *  The default data type is integer - set the toggles.
     */

       arm_tgl( tgl_var_type_int     );
    disarm_tgl( tgl_var_type_real    );
    disarm_tgl( tgl_var_type_string  );
    disarm_tgl( tgl_var_type_unsigned);
    disarm_tgl( tgl_var_type_short   );
    disarm_tgl( tgl_var_type_double  );
}
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    expr_menu.h
*
* FILE FUNCTION:
*
*   This file contains the global variables and function prototypes for the routines
*   in expr_menu.c.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /home/project/3531/Docu/GCB.spec.doc
*
*
* FILE MODULES:
*
*   N/A
*
*****************************************<----->*****************************************/


/*
 *  Constants used in expr_menu.c
 */

#define MAX_UNKNOWN_TOGGLES         5


/*
 *  Global variables used in expr_menu.c
 */

XmString    num_header,
            str_header;

int         active_list[NUM_BUTTONS];

char        buf[300],
            state_buf[2];

char        *b_strings[NUM_BUTTONS] = {
                "and", "or", "not", "bitAnd", "bitOr", "bitXor", "=", "<>", "<=", ">=",
 "<", ">",
                "+",  "-",  "*", "/", "(", ")", ",", "<<", ">>", "PI", "exp", "power",
                "sqrt", "log", "nlog", "sin", "tan", "acos", "asin", "atan"
                };


/*
 *  Function prototypes for routines in expr_menu.c
 */

XtCallbackProc  cbr_cancel(),
                cbr_def_fn(),
                cbr_expr_num(),
                cbr_help(),
                cbr_mat_tgls(),
                cbr_num_input_done(),
                cbr_str_input_done(),
                cbr_unknown_is_mat(),
                cbr_unknown_type_done(),
                cbr_unknown_type_select_name(),
                cbr_unknown_type_select_type(),
                cbr_var_choose(),
                cbr_var_input_done(),
                cbr_var_type(),
                cbr_var_type_done();

void            init_inputs(),
                set_active();


/*
 *  Callback lists for building the selection boxes.  The selection boxes
 *  accept Callback lists instead of Callback Procs.
 */

XtCallbackRec   var_input_done_code[] = {
    { (XtCallbackProc) cbr_var_input_done,  (caddr_t) NULL },
    { (XtCallbackProc) NULL,                (caddr_t) NULL }
    };

XtCallbackRec   var_input_cancel_code[] = {
    { (XtCallbackProc) cbr_var_input_done,  (caddr_t) NULL },
    { (XtCallbackProc) NULL,                (caddr_t) NULL }
    };

XtCallbackRec   var_selc_code[] = {
    { (XtCallbackProc) cbr_var_choose,      (caddr_t) NULL },
    { (XtCallbackProc) NULL,                (caddr_t) NULL }
    };

XtCallbackRec   def_fn_done_code[] = {
    { (XtCallbackProc) cbr_def_fn,          (caddr_t) 0 },
    { (XtCallbackProc) NULL,                (caddr_t) NULL }
    };

XtCallbackRec   def_fn_cancel_code[] = {
    { (XtCallbackProc) cbr_cancel,          (caddr_t) DEF_FN_CANCEL },
    { (XtCallbackProc) NULL,                (caddr_t) NULL }
    };

XtCallbackRec   def_fn_help_code[] = {
    { (XtCallbackProc) cbr_help,            (caddr_t) DEF_FN_HELP },
    { (XtCallbackProc) NULL,                (caddr_t) NULL }
    };
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    fonts.h
*
* FILE FUNCTION:
*
*   This file contains the variables used to maintain the X Windows fonts.  These
*   global variables are maintained in this file so every source file does not need
*   to include the X font header files.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /home/project/3531/Docu/GCB.spec.doc
*
*
* FILE MODULES:
*
*   N/A
*
*****************************************<----->*****************************************/

/*
 *  Font variables.
 */

Font    WAFont,
        big_font, small_font, teeny_font;
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    gcb.c
*
* FILE FUNCTION:
*
*   This file contains the main() routine which makes the calls to start X windows
*   and to build the MOTIF-based user inteface.  This file also contains the signal
*   handler routine which is installed to catch core dump signals.
*
*
* FILE MODULES:
*
*   handler()   - core dump signal handler routine
*   main()      - main GCB routine, this is where it all starts
*
*****************************************<----->*****************************************/

#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include <signal.h>
#include "gcb.h"
#include "symbol.h"
#include "widgets.h"
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  main()
*
* MODULE FUNCTION:
*
*   This is the main routine which starts the GCB.  This routine makes a call to
*   install the signal handler, initializes X Windows, initializes assorted
*   global variables, then turns control over to X Windows main event loop.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  main( argc, argv )

    int     argc;
    char    **argv;

{
    printf("\nStarting the Graphical Comp Builder  ");

    /*
     *  Install the signal hanlder to catch certain signals, especially those that
     *  catch core dumps.
     */

    signal( SIGBUS,  handler );
    signal( SIGHUP,  handler );
    signal( SIGINT,  handler );
    signal( SIGSEGV, handler );

    ActivePopup = -1;

    /*
     *  Initialize X Windows.  Open the display server connection.
     */

    XtToolkitInitialize();

    app_context = XtCreateApplicationContext();

    display = XtOpenDisplay( app_context, NULL, argv[0], "gcb", NULL, 0, &argc, argv );
    if (! display )
        {
        XtWarning("GCB: Can't open display.");
        exit( ERR );
        }

    /*
     *  Build the MOTIF interface elements.
     */

    init_graphics();

    /*
     *  Init and update global variables and status indicators.
     */
```

```
    init_vars( argc, argv );

    /*
     *  Pass control to X Windows, X will be in control from here until when
     *  the GCB exits.
     */

    XtAppMainLoop( app_context );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  handler()
*
* MODULE FUNCTION:
*
*   This routine is the signal handler routine which is installed to catch core dumps
*   during "Bus errors" and "Segmentation Violations".  This routine is installed by
*   main() to catch these signals.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                                 Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  handler( sig, code, scp )

    int     sig,
            code;
    void    *scp;

{
    /*
     *  Determine which signal we caught, and print a corresponding error
     *  message before exiting.
     */

    switch( sig )
        {
        case SIGHUP  :
        case SIGINT  : printf("GCB: Signal caught, exiting\n");
                       break;
        case SIGBUS  : printf("GCB: Bus error\n");
                       break;
        case SIGSEGV : printf("GCB: Segmentation violation\n");
                       break;
        }

    exit( ERR );
}
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    gcb.h
*
* FILE FUNCTION:
*
*   This file contains the global set of program constants and variables for the
*   Graphical Comp Builder.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /home/project/3531/Docu/GCB.spec.doc
*
*
* FILE MODULES:
*
*   N/A
*
*****************************************<----->*****************************************/


/*
 *  GCB constants.
 */

#define ABORT               -2
#define BUILD_COMP          1
#define BUILD_ELEMENT       2
#define EL_EXT              ".GEF"
#define CMP_EXT             ".CMP"
#define CELL_SIZE           5
#define CELL_ROWS           (MAIN_CANVAS_HEIGHT + MAIN_CANVAS_HEIGHT/2)/5
#define CELL_COLS           (MAIN_CANVAS_WIDTH + MAIN_CANVAS_WIDTH/2) / 5
#define ERR                 -1
#define IGNORE              -2
#define MAIN_CANVAS_HEIGHT  800
#define MAIN_CANVAS_WIDTH   1000
#define MAX_COLORS          13
#define MAX_NAME            50
#define MAX_NODES           50
#define MAX_PATH            100
#define MAX_PURPOSE         500
#define MAX_PURPOSE_DISPLAY 30
#define MAX_TEXT_SYMBOLS    200
#define MAX_SYMBOLS         40
#define MAX_TEXT            300
#define MAX_TEXT_STRINGS    20
#define MAX_UNDECLAREDS     5
#define MIN_BTN_WIDTH       90
#define NULLS               ""
#define NONE                -1
#define NOT_FOUND           -1
#define OK                  0
#define PRINT_REP           0
#define PRINT_REP1          1
#define SUCCESS             0

/*
 *  Modes
 */

#define AddSymbol           1
#define EditSymbol          2
#define LineDraw            3
#define DragSymbol          4
#define EditSymbolContents  5
#define PrintBox            6
#define DeleteBox           7
#define StretchPrintBox     8
#define StretchDeleteBox    9
#define Help                10
#define CopyBox             11
#define StretchCopyBox      12
#define MoveBox             13
#define StretchMoveBox      14
#define GhostCopyBox        15
#define GhostMoveBox        16
#define MatrixMenu          17
#define TrigMenu            18

#define LINE_CELL           1
#define SYMBOL_CELL         2

#define UP                  1
#define DOWN                2
#define RIGHT               3
#define LEFT                4

#define COMP_PURPOSE        0
#define ELEMENT_PURPOSE     1
#define NO_CHANGE           2

#define NO_SHOW             0
#define SHOW                1

/*
 *  Constants used in XCallback routines.
 */

#define DONE                0
#define CANCEL              1
#define HLP                 2
#define LIST_SEL            3
#define MANAGE              4

#define FORM                0
#define LIST                1

/*
 *  Constants to determine the type of Element -> Comp or Library Element.
 */

#define ELEMENT         0
#define LIB             1
#define COMP            2

/*
 *  Bit patterns for the generated status word.
 */

#define NOT_GENERATED       01
#define GENERATED           02

/*
 *  Constants used to define target language of choice.
 */
```

```
#define C                       1
#define MOAL                    2
#define UIL                     3

/*
 *  Constants used in determining matrix dimensions and values.
 */

#define MAT                             2
#define SCAL                            1

/*
 *  Error codes used in calls to error_handler.
 */

#define ERR_SYM_ATTRIBS         1
#define ERR_ADD_SYMBOL          2
#define LIB_ELEM_SYM            3
#define NO_ELEMENT              4

/*
 *  Constants which define keys to identify symbol types.
 */

#define NUM_PALETTE             10
#define BEGIN                   0
#define END                     1
#define IF                      2
#define SET                     3
#define PAUSE                   4
#define GOTO                    5
#define START                   6
#define STOP                    7
#define PRINT                   8
#define TEXT                    9

#define SEGMENT_KEY             50
#define LINE_KEY                51
#define LIST_KEY                52

#define SYMBOL_KEY              100

#define PALETTE_KEY             101
#define TEXT_KEY                111

/*
 *  for linking items with symbols
 */

#define ITEM_KEY                500

/*
 *  for linking text with symbols
 */

#define LTEXT_KEY               200
#define COMP_TEXT_KEY           201
#define COMM_TEXT_KEY           202

/*
 *  for linking element type with symbols
 */

#define ELEM_TYPE_KEY           203
```

```
/*
 *  Data Structures.
 */

/*
 *  Various connecting line structures.
 */

typedef struct
    {
    unsigned int        key;
    int                 arrow_x,
                        arrow_y,
                        orientation,
                        cell_end_x,
                        cell_end_y,
                        cell_start_x,
                        cell_start_y,
                        end_x,
                        end_y,
                        start_x,
                        start_y;

    struct LineSeg      *next,
                        *prev;
    } LineSeg;

typedef struct
    {
    unsigned int        key;
    LineSeg             *line;
    struct Symbol       *from,
                        *to;
    } Line;

typedef struct
    {
    unsigned int        key;
    Line                *line;
    struct LineList     *next,
                        *prev;
    } LineList;

/*
 *  Symbol structure.
 */

typedef struct
    {
    unsigned int        key;
    int                 symbol_type,
                        symbol_generated,
                        cell_height,
                        cell_width,
                        cell_x,
                        cell_y,
                        height,
                        width,
                        ulcx,
                        ulcy;
```

```
    char            *text;
    Line            *next;
    LineList        *from;
    Widget          mycanvas;
    Font            font;

    union   {
            struct
                {
                Line    *false_line,
                        *true_line;
                int     false_x,                /* location of label        */
                        false_y,                /*          "               */
                        true_x,                 /*          "               */
                        true_y;                 /*          "               */
                char    *logical_expr,
                        *comp_expr,
                        *comment;
                } IfSym;
            struct
                {
                char *set_expr;
                } SetSym;
            struct
                {
                int comp_type;
                } ElemSym;
            } Sym;
    } Symbol;

/*
 *  Work Area Cell map structure.
 */

typedef struct
    {
    int     cell_type;
    union   {
            Symbol      *symbol;
            LineList    *lines;
            } cell_entry;
    } Cell;



/*
 *  GCB global variables.
 */


Cell        Cell_Map    [CELL_ROWS][CELL_COLS]; /* Work Area cell map       */
GC          WAgc;                               /* graphics context         */

Symbol      Symbol_Map[MAX_SYMBOLS],
            *Begin_Sym;

Widget      Palette     [NUM_PALETTE],
            current_symbol;

char        Author          [50],               /* author as taken from element file  */
            CompDir         [MAX_NAME],         /* comp directory name with extension */
            CompFile        [MAX_NAME],         /* comp file w/o extension or path    */
            CompPurpose     [MAX_PURPOSE],
            CreateDate      [11],
            Cwd             [MAX_PATH],
            DisplayFile     [MAX_PATH],         /* path and name of Display file      */
            ElementFile     [MAX_NAME],         /* element file w/o extension or path */
            ElementPurpose  [MAX_PURPOSE],
            GCompFile       [MAX_PATH],         /* comp    file with extension & path */
            GElementFile    [MAX_PATH],         /* element file with extension & path */
            DefaultsFile    [MAX_PATH],
            MacrosPath      [MAX_PATH],
            MSIDTable       [MAX_PATH],
            LibPath         [MAX_PATH],
            LogFile         [MAX_PATH],
            PositionPath    [MAX_PATH],         /* position name with extension & path */
            pPosition       [50],               /* position name w/o extension or path */
            Purpose         [MAX_PURPOSE],
            RootElement     [MAX_NAME],
            Swd             [MAX_PATH],         /* GCB executable's directory         */
            UpdateDate      [11],
            UpdateTime      [9],
            UserFuncsPath   [MAX_PATH],         /* path to user defined functions     */
            UserName        [50],               /* user name from /etc/passwd file    */
            WSGlobals       [MAX_PATH],
            *comment_text,                      /*most recent if/set popup comment    */
            *expr_text,                         /*most recent if/set expr popup data  */
            *sym_text;                          /*most recent if/set logic popup data */

int         ActivePopup,
            Audit,                              /*boolean; do audit after each op?    */
            Audited,
            Color,                              /*boolean: is the screen color        */
            colors[MAX_COLORS],                 /*allocated pixel values              */
            ElementType,
            ErrorLogLevel,                      /* 1=normal, 3=debug mode             */
            Mode,                               /*EditSym, AddSym, DragSym, etc       */
            OldMode,                            /* save Work Area mode               */
            SaveNeeded,                         /*boolean                             */
            Snap,                               /*boolean; snap on or off?            */
            TargetLanguage,
            ValidComp,
            ValidElement,
            ValidPosition,
            Zoomed,                             /*boolean; zoomed in or out?          */
            nextsymbol,                         /*index into Symbol_Map next avail sym*/
            curr_sym_type,
            originx,
            originy,
            newx,
            newy,
            LogOrCompText;                      /*is log or comp text being displayed */

unsigned int    LineSegKey,
                LineKey,
                LineListKey,
                SymKey;

XmString    print_header, text_header;

/*
 *  Function prototypes.
 */

void        audit_symbol(),
            build_help(),           /* cbr_help.c                               */
            build_log_popup(),      /* cbr_if.c                                 */
```

```
        cancel_draw(),          /* cbr_cancel.c             */
        cbr_set_anchor(),       /* cbr_menu.c               */
        cbr_startstop_sel(),    /* cbr_printset.c           */
        check_lineseg(),        /* cbr_done.c               */
        clear_cell_map_sym(),   /* symbols.c                */
        edit_text(),            /* edit.c                   */
        elog(),                 /* utils.c                  */
        error_handler(),        /* utils.c                  */
        init_cell_map(),        /* symbols.c                */
        log(),                  /* utils.c                  */
        menu_proc(),
        myinvert(),             /*                          */
        remove_symbol(),        /* symbols.c                */
        set_attribs(),          /* utils.c                  */
        set_current_sym(),      /* symbols.c                */
        set_line_event(),       /* undo.c                   */
        set_midpt(),
        set_symbol_event(),     /* undo.c                   */
        set_sym_attribs(),      /* symbol_table.c           */
        set_user_data(),        /* utils.c                  */
        setup_math_menu(),      /* expr_menu.c              */
        setup_palette(),        /* setup_palette.c          */
        undo(),                 /* undo.c                   */
        update_pos_fields(),
        user_ack(),             /* utils.c                  */
        zoom();                 /* cbr_menu.c               */

int     audit_line(),
        count_qoutes(),         /* cbr_var_input.c          */
        draw_symbol(),          /* images.c                 */
        get_element_calls(),    /* element_file.c           */
        handler(),              /* gcb.c                    */
        init_vars(),            /* init_vars.c              */
        install_symbol(),       /* symbols.c                */
        load_fn_list(),         /* object.c                 */
        next_avail_sym(),       /* symbols.c                */
        update_symbol_pos(),    /* symbols.c                */
        valid_fn_name();        /* object.c                 */
```

```
#define gcb_icon_width 48
#define gcb_icon_height 48
static char gcb_icon_bits[] = {
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x1f, 0x00, 0x00, 0x00, 0x00, 0x80, 0x20, 0x00, 0x00, 0x00, 0x00,
   0x40, 0x40, 0x00, 0x00, 0xfc, 0x7f, 0x20, 0x80, 0x00, 0x00, 0x04, 0x40,
   0x20, 0x80, 0x00, 0x00, 0x04, 0x40, 0xa0, 0xbf, 0x00, 0x80, 0x74, 0x5d,
   0x20, 0x80, 0x00, 0x00, 0x05, 0x40, 0x20, 0x80, 0x00, 0xf0, 0x07, 0x40,
   0x40, 0x40, 0x00, 0x10, 0xb5, 0x5b, 0x80, 0x20, 0x00, 0x90, 0x04, 0x40,
   0x00, 0x1f, 0x00, 0x10, 0x04, 0x40, 0x00, 0x04, 0x00, 0x10, 0x04, 0x40,
   0x00, 0x04, 0x00, 0x10, 0xfc, 0x7f, 0x00, 0x04, 0x00, 0x10, 0x00, 0x02,
   0x00, 0x04, 0x00, 0x10, 0x00, 0x02, 0x00, 0x04, 0x00, 0x10, 0x00, 0x02,
   0x00, 0x04, 0x00, 0x10, 0x00, 0x02, 0x00, 0x04, 0x00, 0x10, 0x00, 0x02,
   0x00, 0x04, 0x00, 0x10, 0x00, 0x02, 0x00, 0x15, 0x00, 0x10, 0x00, 0x02,
   0x00, 0x0e, 0x00, 0x10, 0x00, 0x02, 0x00, 0x04, 0x00, 0x10, 0x80, 0x0a,
   0xe0, 0xff, 0x00, 0x10, 0x00, 0x07, 0x10, 0x00, 0x01, 0x10, 0x00, 0x02,
   0x08, 0x00, 0x02, 0x10, 0x80, 0x0f, 0xe4, 0xff, 0x04, 0x10, 0x40, 0x10,
   0x02, 0x00, 0x08, 0x10, 0x24, 0x20, 0x01, 0x00, 0x10, 0x10, 0x28, 0x20,
   0x01, 0x0f, 0xf0, 0x1f, 0x3f, 0x27, 0x01, 0x00, 0x10, 0x00, 0x29, 0x20,
   0x02, 0x00, 0x08, 0x00, 0x25, 0x20, 0xe4, 0xff, 0x04, 0x00, 0x41, 0x10,
   0x08, 0x00, 0x02, 0x00, 0x81, 0x0f, 0x10, 0x00, 0x01, 0x00, 0x01, 0x00,
   0xe0, 0xff, 0x00, 0x00, 0x01, 0x00, 0x00, 0x04, 0x00, 0x00, 0x01, 0x00,
   0x00, 0x04, 0x00, 0x00, 0x01, 0x00, 0x00, 0x04, 0x00, 0x00, 0x01, 0x00,
   0x00, 0x04, 0x00, 0x00, 0x01, 0x00, 0x00, 0x04, 0x00, 0x00, 0x01, 0x00,
   0x00, 0x04, 0x00, 0x00, 0x01, 0x00, 0x00, 0xfc, 0xff, 0xff, 0x01, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    gcb_parse.h
*
* FILE FUNCTION:
*
*   This file contains constants and variables which are utilized to parse an expression
.
*   The YACC defined grammar allows us to determine if a statement is syntactically
*   correct but addtional work must be performed to review the semantics of the expressi
on.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

/*
 *  Defines used for return codes from the parsing routines.
 */

#define PARSE_SUCCESS       00
#define NO_PARSE_MEMORY     01
#define UNDECLARED          02
#define SYNTAX_ERR          04
#define END_OF_FILE         05

#define LHS 0
#define RHS 1

/*
 *  Function prototypes.
 */

void set_state();

/*
 *  Parsing variables.
 */

int paren_count, WhereAmI, SetSym, stable_state;

/*
 *  Constants used when performing type checking on expressions.
 */

#define INVALID_EXPRESSION      0
#define VALID_EXPRESSION        1

/*
 *  Constants used to indicate whether or not a function call is in progress.
 */

#define ON                      1
#define OFF                     0

/*
 *  Constants used to indicate whether or not a function is a user defined or GCB
 *  defined function.
 */

#define USER_DEFINED            01
#define GCB_DEFINED             02

/*
 *  Various defines.
 */

#define ADD_OPER                01
#define MINUS_OPER              02
#define MULT_OPER               04
#define DIV_OPER                010
#define IDENT_OPER              020
#define INVER_OPER              040
#define TRANS_OPER              0100
#define CROSS_OPER              0200
#define DOT_OPER                0400
#define SHIFT_OPER              01000
#define MADD_OPER               02000
#define MMINUS_OPER             04000
#define MMULT_OPER              010000

#define MATRIX_OPERATION        01
#define SCALAR_OPERATION        02

#define MAX_PARSE_MESSAGE       80

/*
 *  Data structure to hold matrix data.
 */

struct matrix_data {
    char    md_name[ MAX_SYMBOL_SIZE + 1 ];
    int     md_attributes;
    short   md_num_dimensions;
    short   md_subs[ MAX_DIMENSIONS ];
};
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    images.c
*
*
* FILE FUNCTION:
*
*   This file contains the routines which draw the symbols into the symbol
*   canvases.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /home/project/3531/Docu/GCB.spec.doc
*
*
* FILE MODULES:
*
*   char_height()        - determines the height of each char given the current font.
*   chars_per_line()     - determines # of chars per line given char size and sym width.
*   determine_height()   - sets the height of the symbol based on the text.
*   draw_begin()         - draws the begin symbol.
*   draw_end()           - draws the end symbol.
*   draw_goto()          - draws the goto symbol.
*   draw_if()            - draws the if symbol.
*   draw_pause()         - draws the pause symbol.
*   draw_print()         - draws the print symbol.
*   draw_set()           - draws the set symbol.
*   draw_start()         - draws the start symbol.
*   draw_stop()          - draws the stop symbol.
*   draw_symbol()        - depending on parameter, draws the correct symbol.
*   draw_text()          - draws text into a print, if, or set symbol.
*   draw_text_symbol()   - draws the text symbol.
*   num_Lines()          - determines # lines a string will need given a width.
*   longest_line()       - determines the longest line in a string.
*   stringcpy()          - copies 1 string, starting at char n, to parameter len
*   x_offset()           - Determines horizontal offset to center string in symbol.
*
*****************************************<----->*****************************************/

#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>

#include "gcb.h"
#include "lines.h"
#include "widgets.h"
#include "images.h"
#include "fonts.h"

/*****************************************<----->*****************************************
*
* MODULE NAME:  char_height()
*
*
* MODULE FUNCTION:
*
*   This routine returns the height of each char given the current font.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  char_height()

{
    if ( WAFont == big_font )
        return( 15 );
    else if ( WAFont == small_font )
        return( 12 );
    else if ( WAFont == teeny_font )
        return( 10 );
}
```

```
/*****************************************<-----*****************************************
*
* MODULE NAME:  chars_per_line()
*
* MODULE FUNCTION:
*
*   This routine determines how many characters will fit on a line of a given
*   width given the current font.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.01 - 08/01/91
*                            Release 1.02 - 08/28/91
*
*****************************************<-----*****************************************/

int  chars_per_line( w )

    int w;

{
    if ( WAFont == big_font )
        return( (w-7) / 9 );
    else if ( WAFont == small_font )
        return( (w-7) / 6 );
    else
        return( (w-7) / 6 );
}
```

```
/*****************************************<-----*****************************************
*
* MODULE NAME:  determine_height()
*
* MODULE FUNCTION:
*
*   This routine uses the width of the symbol, the font, and the amount of text
*   to determine the symbol's height.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.01 - 08/01/91
*                            Release 1.02 - 08/28/91
*
*****************************************<-----*****************************************/

int  determine_height( widget, w )

    Widget  widget;
    int     w;

{
    Arg     args[1];
    int     expr_lines,
            sym_lines,
            sym_type;

    if ( (!sym_text) && (!expr_text) )
        {
        user_ack("determine height: no logical or expression text, empty symbol");
        cancel_draw();
        return( ERR );
        }

    /*
     *  determine height of symbol depending on the longer of the 2 texts.
     */

    sym_lines = numlines( sym_text , chars_per_line(w) );

    /*
     *  This routine is called only for PRINT, SET, and IF symbols;
     *  print symbol height is always determined by sym_text, if/set height
     *  by longer of expr_text and sym_text.
     */

    XtSetArg( args[0], XmNuserData, &sym_type );
    XtGetValues( widget, args, 1 );

    if ( sym_type == PRINT )
        {
        if ( Zoomed )
            XtSetArg( args[0], XmNheight, sym_lines*char_height()+31 );
        else XtSetArg( args[0], XmNheight, sym_lines*char_height()+46 );
        }
    else
        {
        expr_lines = numlines( expr_text, chars_per_line(w) );
        if ( sym_lines < expr_lines )
            sym_lines = expr_lines;
```

```
        /*
         *  SET must be at least 2 lines or it looks silly.
         */

        if ( (sym_type == SET) && (sym_lines < 2) )
            sym_lines = 2;

        /*
         *  set the height of if symbols based on # of lines plus space for
         *  shape, depending on whether or not we are zoomed; height of set
         *  is constant 10 plus the number of lines.
         */

        if ( Zoomed )
             XtSetArg( args[0], XmNheight, sym_lines*char_height() +
                ((sym_type==IF) ? 31 : 10) );
        else XtSetArg( args[0], XmNheight, sym_lines*char_height() +
                ((sym_type==IF) ? 46 : 10) );
        }
    XtSetValues( widget, args, 1 );

    return( sym_lines );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  draw_begin()
*
*
* MODULE FUNCTION:
*
*   This routine draws the begin symbol.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

void  draw_begin( gc, w, h )

    GC  gc;
    int w, h;
{
    int x_val = x_offset( w, "BEGIN" );

    /*
     *  draw a filled oval in the begin symbol's colors.
     */

    XSetForeground( display, gc, colors[get_my_background(BEGIN)] );

    XFillArc( display, drawable, gc, 3, 0, w-5, h-3,  90*64,  90*64 );
    XFillArc( display, drawable, gc, 3, 0, w-5, h-3,  90*64, -90*64 );
    XFillArc( display, drawable, gc, 3, 0, w-5, h-3, 270*64, -90*64 );
    XFillArc( display, drawable, gc, 3, 0, w-5, h-3, 270*64,  90*64 );

    XSetForeground( display, gc, colors[get_my_foreground(BEGIN)] );

    /*
     *  draw an oval, then the word "BEGIN"
     */

    h = h-3;
    XDrawArc( display, drawable, gc, xx, 0, w - 5, h,  90*64,  90*64 );
    XDrawArc( display, drawable, gc, xx, 0, w - 5, h,  90*64, -90*64 );
    XDrawArc( display, drawable, gc, xx, 0, w - 5, h, 270*64,  90*64 );
    XDrawArc( display, drawable, gc, xx, 0, w - 5, h, 270*64, -90*64 );

    XDrawString( display, drawable, gc, x_val, h/2 + 4, "BEGIN", 5 );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  draw_end()
*
*
* MODULE FUNCTION:
*
*   This routine draws the end symbol.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

void  draw_end( gc, w, h )

    GC  gc;
    int w, h;

{
    int x_val = x_offset( w, "END" );

    /*
     *  make a square out of the parameter rectangle - thus the arcs draw a circle
     */

    if ( w > h )
        w = h;
    else if ( h > w )
        h = w;

    /*
     *  draw a filled oval in the begin symbol's colors.
     */

    XSetForeground( display, gc, colors[get_my_background(END)] );

    XFillArc( display, drawable, gc, 3, 3, w - 3, h - 5,  90*64,  90*64 );
    XFillArc( display, drawable, gc, 3, 3, w - 3, h - 5,  90*64, -90*64 );
    XFillArc( display, drawable, gc, 3, 3, w - 3, h - 5, 270*64,  90*64 );
    XFillArc( display, drawable, gc, 3, 3, w - 3, h - 5, 270*64, -90*64 );

    XSetForeground(display, gc, colors[get_my_foreground(END)] );

    /*
     *  draw a circle, then the word END.
     */

    XDrawArc( display, drawable, gc, xx, yy, w - 3, h - 5,  90*64,  90*64 );
    XDrawArc( display, drawable, gc, xx, yy, w - 3, h - 5,  90*64, -90*64 );
    XDrawArc( display, drawable, gc, xx, yy, w - 3, h - 5, 270*64,  90*64 );
    XDrawArc( display, drawable, gc, xx, yy, w - 3, h - 5, 270*64, -90*64 );

    XDrawString( display, drawable, gc, x_val, h/2 + 4, "END", 3 );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  draw_goto()
*
*
* MODULE FUNCTION:
*
*   This routine draws the goto symbol.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

void  draw_goto( canvas, gc, w, h )

    GC      gc;
    int     w, h;
    Widget  canvas;

{
    int x_val = x_offset( w, sym_text );

    h = h+5;

    /*
     *  draw a filled semicircle on either side of a filled rectangle.
     */

    XSetForeground( display, gc, colors[get_my_background(GOTO)] );

    XFillArc( display, drawable, gc, 3, -5, w - 5, h, 135*64,  90*64 );
    XFillArc( display, drawable, gc, 3, -5, w - 5, h,  45*64, -90*64 );

    if ( Zoomed )
        XFillRectangle( display, drawable, gc, 11, 4, 56, h-14 );
    else
        XFillRectangle( display, drawable, gc, 17, 6, 81, h-21 );

    XSetForeground( display, gc, colors[get_my_foreground(GOTO)] );

    /*
     *  draw a semicircle on either side of a rectangle; draw inside the
     *  name of the element to go to.
     */

    XDrawArc( display, drawable, gc, xx, -5, w - 5, h, 135*64,  90*64 );
    XDrawArc( display, drawable, gc, xx, -5, w - 5, h,  45*64, -90*64 );

    if ( Zoomed )
        {
        XDrawLine( display, drawable, gc, 12, 3, 64, 3 );
        XDrawLine( display, drawable, gc, 12, h - 11, 64, h - 11 );

        XDrawString( display, drawable, gc, 30 , h/2 - 6, "CALL", 4 );
        XDrawString( display, drawable, gc,x_val,h/2+6,sym_text, strlen(sym_text) );
        }
    else
        {
```

```
        XDrawLine( display, drawable, gc, 18, 6, 95, 6 );
        XDrawLine( display, drawable, gc, 18, h - 15, 95, h - 15 );

        XDrawString( display, drawable, gc, 45, h/2 - 4, "CALL", 4 );
        XDrawString( display, drawable, gc, x_val, 50, sym_text, strlen(sym_text) );
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  draw_if()
*
*
* MODULE FUNCTION:
*
*   This routine draws the if symbol.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                                  Release 1.01 - 08/01/91
*                                  Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  draw_if( canvas, gc, w, h )

    GC      gc;
    int     w, h;
    Widget  canvas;

{
    Arg         args[1];
    Dimension   nh;
    int         lines, new_height;

    /*
     *  set height based on number of lines of text
     */

    if ( (lines = determine_height(canvas, w)) == ERR )
        return;

    XtSetArg( args[0], XmNheight, &nh );
    XtGetValues( canvas, args, 1 );

    /*
     *  color symbol and draw text.
     */

    if ( draw_text(gc,0,0,(int) nh,w,IF) == ERR )
        return;

    new_height = (int) nh;

    /*
     *  draw top triangle
     */

    XDrawLine( display, drawable, gc, 3, Zoomed ? 16 : 23, w/2, 3 );
    XDrawLine( display, drawable, gc, w/2, 3, w-1, Zoomed ? 16 : 23 );

    /*
     *  draw bottom triangle
     */

    XDrawLine( display, drawable, gc, 3, Zoomed ? new_height - 16 : new_height-23,
        w/2, new_height-3 );
    XDrawLine( display, drawable, gc, w/2, new_height-3,
        w-1, Zoomed ? new_height-16 : new_height-23 );
```

```
    /*
     *  draw sides
     */

    if ( Zoomed )
        {
        XDrawLine( display, drawable, gc, 3, 16, 3, new_height-16 );
        XDrawLine( display, drawable, gc, w-1, 16, w-1, new_height-16 );
        }
    else
        {
        XDrawLine( display, drawable, gc, 3, 23, 3, new_height-23 );
        XDrawLine( display, drawable, gc, w-1, 23, w-1, new_height-23 );
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  draw_pause()
*
*
* MODULE FUNCTION:
*
*   This routine draws the pause symbol.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

void  draw_pause( canvas, gc, w, h )

    GC      gc;
    int     w, h;
    Widget  canvas;

{
    int x_val = x_offset( w, sym_text );

    /*
     *  make a square out of the parameter rectangle - thus the arcs draw a circle
     */

    if ( w > h )
        w = h;
    else if ( h > w )
        h = w;

    /*
     *  draw a filled circle in the pause symbol's colors.
     */

    XSetForeground( display, gc, colors[get_my_background(PAUSE)] );

    XFillArc( display, drawable, gc, 3, 3, w - 3, h - 5,  90*64,  90*64 );
    XFillArc( display, drawable, gc, 3, 3, w - 3, h - 5,  90*64, -90*64 );
    XFillArc( display, drawable, gc, 3, 3, w - 3, h - 5, 270*64,  90*64 );
    XFillArc( display, drawable, gc, 3, 3, w - 3, h - 5, 270*64, -90*64 );

    XSetForeground(display, gc, colors[get_my_foreground(PAUSE)] );

    /*
     *  draw a circle.
     */

    XDrawArc( display, drawable, gc, xx, yy, w - 3, h - 5,  90*64,  90*64 );
    XDrawArc( display, drawable, gc, xx, yy, w - 3, h - 5,  90*64, -90*64 );
    XDrawArc( display, drawable, gc, xx, yy, w - 3, h - 5, 270*64,  90*64 );
    XDrawArc( display, drawable, gc, xx, yy, w - 3, h - 5, 270*64, -90*64 );

    /*
     *  draw the pause amount, then the clock face numerals in smaller font.
     */

    XDrawString( display, drawable, gc, x_val, h/2+4, sym_text, strlen(sym_text) );
```

```
    if (! Zoomed )
        XSetFont( display, WAgc, (Font)teeny_font );

    XDrawString( display, drawable, gc, 5, h/2, "9", 1 );
    XDrawString( display, drawable, gc, w-8, h/2, "3", 1 );
    XDrawString( display, drawable, gc, w/2-2, 10, "12", 2 );
    XDrawString( display, drawable, gc, w/2, h-5, "6", 1 );

    XDrawLine( display, drawable, gc, w/2, 12, w/2, h/2-9 );
    XDrawLine( display, drawable, gc, w/2, h/2+8, w/4, h-9 );

    /*
     *  restore former font
     */

    if (! Zoomed )
        XSetFont( display, WAgc, (Font) WAFont );
}
```

```
/*****************************************<----->*************************************
*
* MODULE NAME:  draw_print()
*
*
* MODULE FUNCTION:
*
*   This routine draws the print symbol.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                                  Release 1.01 - 08/01/91
*                                  Release 1.02 - 08/28/91
*
*****************************************<----->*************************************/

void  draw_print( canvas, gc, w, h )

    Widget  canvas;
    GC      gc;
    int     w, h;

{
    Arg         args[1];
    Dimension   nh;
    int         lines, new_height;

    /*
     *  set height based on number of lines of text
     */

    if ( (lines = determine_height(canvas, w)) == ERR )
        return;

    XtSetArg( args[0], XmNheight, &nh );
    XtGetValues( canvas, args, 1 );

    /*
     *  color symbol and draw text.
     */

    if ( draw_text(gc, 0, 0, (int)nh, w, PRINT) == ERR )
        return;

    new_height = (int)nh;

    /*
     *  draw top lines
     */

    if ( Zoomed )
        {
        XDrawLine( display, drawable, gc, 3, 16, 37, 3 );
        XDrawLine( display, drawable, gc, 37, 3, w-3, 3 );
        }
    else
        {
        XDrawLine( display, drawable, gc, 3, 23, 56, 3 );
        XDrawLine( display, drawable, gc, 56, 3, w-3, 3 );
        }
```

```
    /*
     *  draw bottom line
     */

    XDrawLine( display, drawable, gc, 3, new_height-3, w-3, new_height-3 );

    /*
     *  draw sides
     */

    XDrawLine( display, drawable, gc, 3, (Zoomed ? 16 : 23), 3, new_height-3 );
    XDrawLine( display, drawable, gc, w-3, 3, w-3, new_height-3 );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  draw_set()
*
*
* MODULE FUNCTION:
*
*   This routine draws the set symbol.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                                  Release 1.01 - 08/01/91
*                                  Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

void  draw_set( canvas, gc, w, h )

    GC      gc;
    Widget  canvas;
    int     w, h;

{
    Arg         args[1];
    int         lines, new_height;
    Dimension   nh;

    /*
     *  set height based on number of lines of text
     */

    if ( (lines = determine_height(canvas, w)) == ERR )
        return;

    XtSetArg( args[0], XmNheight, &nh );
    XtGetValues( canvas, args, 1 );

    /*
     *  color symbol and draw text.
     */

    if ( draw_text(gc, 0, 0, (int)nh, w, SET) == ERR )
        return;

    new_height = (int)nh;

    /*
     *  draw top line
     */

    XDrawLine( display, drawable, gc, 3, 3, w-1, 3 );

    /*
     *  draw bottom line
     */

    XDrawLine( display, drawable, gc, 3, new_height-1, w-3,new_height-1 );

    /*
     *  draw sides
```

```
    */

    XDrawLine( display, drawable, gc, 1, 3, 1, new_height-3 );
    XDrawLine( display, drawable, gc, w-1, 3, w-1, new_height-3 );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  draw_start()
*
*
* MODULE FUNCTION:
*
*   This routine draws the start symbol.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

void  draw_start( canvas, gc, w, h )

    GC      gc;
    Widget  canvas;
    int     w, h;

{
    int x_val = x_offset( w, sym_text );

    /*
     *  draw a filled rectangle.
     */

    XSetForeground( display, gc, colors[get_my_background(START)] );
    XFillRectangle( display, drawable, gc, 3, 3, w-5, h-5 );
    XSetForeground( display, gc, colors[get_my_foreground(START)] );

    /*
     *  draw a rectangle, then the vertical lines that distinguish a start symbol,
     *  then the name of the element to start.
     */

    XDrawRectangle( display, drawable, gc, xx, yy, w-5, h-5 );

    if ( Zoomed )
        {
        XDrawLine( display, drawable, gc, xx + 3, yy, xx + 3, h - 4 );
        XDrawLine( display, drawable, gc, w - (xx + 2), yy, w - (xx + 2), h - 4 );

        XDrawString( display, drawable, gc, 14, h/2 - 2, "ACTIVATE", 8 );
        XDrawString( display, drawable, gc,x_val,h/2+6,sym_text, strlen(sym_text) );
        }
    else
        {
        XDrawLine( display, drawable, gc, xx + 10, yy, xx + 10, h - 4 );
        XDrawLine( display, drawable, gc, w - 12, yy, w - 12, h - 4 );

        XDrawString( display, drawable, gc, w/2 - (XTextWidth(fs, "ACTIVATE", 8)/2),
            h/2 - 9, "ACTIVATE", 8 );
        XDrawString( display, drawable, gc, x_val, 42, sym_text, strlen(sym_text) );
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  draw_stop()
*
*
* MODULE FUNCTION:
*
*   This routine draws the stop symbol.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

void  draw_stop( canvas, gc, w, h )

    GC      gc;
    Widget  canvas;
    int     w, h;

{
    int     x_val = x_offset( w, sym_text );
    XPoint  points[8];

    /*
     *  fill an octagon.
     */

    XSetForeground( display, gc, colors[get_my_background(STOP)] );

    points[0].x = Zoomed ?  1 : 2;   points[0].y = Zoomed ? 19 : 28;
    points[1].x = Zoomed ? 20 : 30;  points[1].y = Zoomed ?  2 : 3;
    points[2].x = Zoomed ? 43 : 65;  points[2].y = Zoomed ?  2 : 3;
    points[3].x = Zoomed ? 60 : 90;  points[3].y = Zoomed ? 19 : 28;
    points[4].x = Zoomed ? 60 : 90;  points[4].y = Zoomed ? 42 : 63;
    points[5].x = Zoomed ? 43 : 65;  points[5].y = Zoomed ? 59 : 88;
    points[6].x = Zoomed ? 20 : 30;  points[6].y = Zoomed ? 59 : 88;
    points[7].x = Zoomed ?  1 : 2;   points[7].y = Zoomed ? 42 : 63;

    XFillPolygon( display, drawable, gc, points, 8, Convex, CoordModeOrigin );
    XSetForeground( display, gc, colors[get_my_foreground(STOP)] );

    /*
     *  draw an octagon.
     */

    if ( Zoomed )
        {
        XDrawLine( display, drawable, gc, 1, 19, 1, 42 );
        XDrawLine( display, drawable, gc, 1, 19, 20, 2 );
        XDrawLine( display, drawable, gc, 20, 2, 43, 2 );
        XDrawLine( display, drawable, gc, 43, 2, 59, 19 );
        XDrawLine( display, drawable, gc, 59, 19, 59, 42 );
        XDrawLine( display, drawable, gc, 43, 59, 59, 42 );
        XDrawLine( display, drawable, gc, 20, 59, 43, 59 );
        XDrawLine( display, drawable, gc, 1, 42, 20, 59 );

        XDrawString( display, drawable, gc, 19, h/2 - 4, "STOP", 4 );
        XDrawString( display, drawable, gc,x_val,h/2+4,sym_text, strlen(sym_text));
        }
    else
        {
        XDrawLine( display, drawable, gc,  2, 28,  2, 63 );
        XDrawLine( display, drawable, gc,  2, 28, 30,  3 );
        XDrawLine( display, drawable, gc, 30,  3, 65,  3 );
        XDrawLine( display, drawable, gc, 65,  3, 88, 28 );
        XDrawLine( display, drawable, gc, 88, 28, 88, 63 );
        XDrawLine( display, drawable, gc, 65, 88, 88, 63 );
        XDrawLine( display, drawable, gc, 30, 88, 65, 88 );
        XDrawLine( display, drawable, gc,  2, 63, 30, 88 );

        XDrawString( display, drawable, gc, w/2 - (XTextWidth(fs, "STOP", 4)/2),
            h/2 - 8, "STOP", 4 );
        XDrawString( display, drawable, gc,x_val,h/2+8,sym_text, strlen(sym_text) );
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  draw_symbol()
*
*
* MODULE FUNCTION:
*
*   This routine calls the routine which draws the specified symbol image
*   into the specified symbol canvas.
*
*       for if/set, the basic algorithm is:
*
*       determine_height
*       fill in outline
*       draw_text
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

int  draw_symbol( symbol_type, canvas, gc, font )

    int     symbol_type;
    Widget  canvas;
    GC      gc;
    Font    font;

{
    int                 w, h;
    XWindowAttributes   attribs;

    /*
     *  use XLib calls to get the drawable resouce of the parameter window and
     *  its size and to clear the window of its old contents.
     */

    drawable = XtWindow( canvas );
    if (! XGetWindowAttributes(display,drawable,&attribs) )
        error_handler( ERR_SYM_ATTRIBS, "draw_symbol" );

    w = attribs.width;
    h = attribs.height;

    XClearWindow( display, drawable );

    if ( font )
        XSetFont( display, WAgc, WAFont = (Font)font );

    /*
     *  get the font currently being used in the parameter gc; set variable
     *  fs to this font.  All draw requests will be made using this font.
     */

    gcon = XGContextFromGC( gc );
    fs = XQueryFont( display, gcon );

    switch ( symbol_type )
        {
        case BEGIN:     { draw_begin( gc, w, h ); break; }
        case END:       { draw_end( gc, w, h ); break; }
        case GOTO:      { draw_goto( drawable, gc, w, h );      break; }
        case START:     { draw_start( canvas, gc, w, h );       break; }
        case STOP:      { draw_stop( canvas, gc, w, h );        break; }
        case PAUSE:     { draw_pause( canvas, gc, w, h );       break; }
        case IF:        { draw_if( canvas, gc, w, h );          break; }
        case PRINT:     { draw_print( canvas, gc, w, h );       break; }
        case SET:       { draw_set( canvas, gc, w, h );         break; }
        case TEXT:      { draw_text_symbol( canvas, gc, w, h ); break; }
        default:        { user_ack("unknown symbol_type");      exit(1); }
        }

    /*
     *  default foreground color is black
     */

    XSetForeground( display, WAgc, BlackPixel(display, DefaultScreen(display)) );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  draw_text()
*
*
* MODULE FUNCTION:
*
*   This routine determines the size of variable-size symbols based on the amount
*   of text they contain; it then colors the symbol and prints the text. The
*   symbol must be colored first since the 'fill' routines clear the previous
*   contents.
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.01 - 08/01/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

int  draw_text( gc, x, y, h, w, type )

    GC  gc;
    int x, y, h, w, type;

{
    int     i = 0, j = 0,
            yoffset,
            which_text,
            num_chars;
    char    temp[MAX_TEXT];
    XPoint  points[6];

    /*
     *  fill in symbol outline. Do this before drawing text since  XFill*
     *  routines clear the symbol.
     */

    XSetForeground( display, gc, colors[get_my_background(type)] );
    switch( type )
        {
        case IF:

            points[0].x =  3;           points[0].y =  Zoomed ? 16 : 23;
            points[1].x = w/2;          points[1].y = 3;
            points[2].x = w;            points[2].y = Zoomed ? 16 : 23;
            points[3].x = w;            points[3].y = Zoomed ? h-16 : h-23;
            points[4].x = w/2;          points[4].y = h-3;
            points[5].x = 3;            points[5].y = Zoomed ? h-16 : h-23;

            XFillPolygon( display, drawable, gc, points, 6, Convex, CoordModeOrigin );
            break;

        case PRINT:

            points[0].x =  3;               points[0].y =  Zoomed ? 16 : 23;
            points[1].x = Zoomed ? 37 : 56; points[1].y = 3;
            points[2].x = w;                points[2].y = 3;
            points[3].x = w;                points[3].y = h;
            points[4].x = 3;                points[4].y = h;

            XFillPolygon( display, drawable, gc, points, 5, Convex, CoordModeOrigin );
            break;

        case SET:

            XFillRectangle( display, drawable, gc, 0, 3, w, h );
            break;
        }

    XSetForeground( display, gc, colors[get_my_foreground(type)] );

    /*
     *  set vertical offset into symbol.
     */

    if ( type == SET )
        yoffset = char_height();
    else
        {
        if ( Zoomed )
            yoffset = 18 + char_height();
        else
            yoffset = 22 + char_height();
        }

    /*
     *  which text are we going to display - logical or expression
     */

    if ( (type == PRINT) || (!LogOrCompText) )
        which_text = 1;
    else
        which_text = 0;

    /*
     *  determine length of either logical or expr text; if none, length is 0.
     */

    if ( which_text )
        {
        if ( sym_text )
            num_chars = strlen( sym_text );
        else
            num_chars = 0;
        }
    else
        {
        if ( expr_text )
            num_chars = strlen( expr_text );
        else
            num_chars = 0;
        }

    /*
     *  copy segments of text into buffer, and draw them on the screen,
     *  yoffset apart.
     */

    while ( (i += stringcpy(temp, which_text ? sym_text : expr_text,i,chars_per_line(w)))

        <= num_chars )
        {
        if ( Zoomed )
            XDrawString( display, drawable, gc, x ? x : 5,
                y + (j*char_height()) + yoffset,
```

```
        temp, strlen(temp) );
    else XDrawString( display, drawable, gc, x ? x : 5,
        y + (j*char_height()) + yoffset,
        temp, strlen(temp) );
    ++j;
    }
}
```

```
/*************************************<----->*************************************
*
* MODULE NAME:  draw_text_symbol()
*
*
* MODULE FUNCTION:
*
*   This routine draws the text symbol.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*************************************<----->*************************************/

void  draw_text_symbol( canvas, gc, w, h )

    Widget  canvas;
    GC      gc;
    int     w, h;
{
    Arg     args[10];
    char    *line;
    int     lines,
            chars_per_line;

    /*
     *  Set the width of the canvas to fit the longest line of text.
     */

    line = longest_line( sym_text, &chars_per_line );

    XtSetArg( args[0], XmNwidth, XTextWidth(fs, line, chars_per_line) + 3 );
    XtSetValues( canvas, args, 1 );

    /*
     *  Set the height to fit the # of lines X the char height
     */

    lines = numlines( sym_text, chars_per_line );
    XtSetArg( args[0], XmNheight, lines * (char_height()+5) );
    XtSetValues( canvas, args, 1 );

    XSetForeground( display, gc, colors[get_my_foreground(TEXT)] );
    set_text( canvas, gc, 2, char_height(), sym_text );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  numlines()
*               longest_line()
*
*
* MODULE FUNCTION:
*
*   numlines:   determines the number of lines the string parameter will require,
*               given the width of the symbol
*
*   longest_line : determines the maximum width necessary for the symbol.
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

int  numlines( buf, chars_per )

    char    *buf;
    int     chars_per;
{
    char    *p = buf;
    int     i = 0, count = 0, placeholder = 0, reset = 1, lines = 1;

    if ( !buf )

        /*
         *  NULL string requires 1 line.
         */

        return( lines );

    /*
     *  skip til we find the end of the current word, a newline, or the
     *  end of buf.
     */

    while ( TRUE )
        {
        while ( (p[i]) && (p[i] != ' ') && (p[i] != '\n') && (count < chars_per) )
            {
            i++;
            count++;
            }
        if ( (p[i] == ' ') || (!p[i]) )
            {

            /*
             *  end of word or end of buf
             */

            if ( !p[i] )

                /*
                 *  done
                 */

                return( lines );

            else
                {

                /*
                 *  find start of next word
                 */

                while ( p[i] == ' ' )
                    {
                    i++;
                    count++;
                    }
                placeholder = i;
                reset = 0;
                }
            }
        else if ( p[i] == '\n' )
            {

            /*
             *  ++ lines at newline
             */

            lines++;
            i++;
            placeholder = i;
            reset = 1;
            count = 0;
            }
        else if ( count >= chars_per )
            {

            /*
             *  1 long word; ++lines, reset count for new line.
             */

            lines++;
            count = 0;

            if ( reset )
                placeholder = i;
            else
                {
                i = placeholder;
                reset = 1;
                }

            }
        }
}


char  *longest_line( buf, w )

    char    *buf;
    int     *w;
{
    char *temp, *temp2;
    char *p = buf;
    int longest = 0;

    /*
     *  count # of chars until newline or EOF; longest span between
     *  beginning, newline, and EOF is longest line.
```

```
     */

    while ( *p )
        {
        int i = 0;
        temp = p;
        while ( (*p) && (*p != '\n') )
            {
            p++;
            i++;
            }
        if ( i > longest )
            {
            *w = longest = i;
            temp2 = temp;
            }
        if ( *p == '\n' )

            /*
             *  pass over newline; reset count
             */

            p++;
        }
    return( (char *) temp2 );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME stringcpy()
*
*
* MODULE FUNCTION:
*
*   This routine copies string t to string s starting at char start, up to size or
*   null byte, whichever comes first. returns number of chars copied.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

int  stringcpy( s, t, start, size )

    char    *s, *t;
    int     start, size;
{
    char *p = s;
    char *q = t + start;
    int  j, i = 0, total = 0;

    if ( (!q) || (!*q) )
        return( 1 );

    while ( TRUE )
        {
        i = 0;
        while ( (q[i]) && (q[i] != ' ') && (q[i] != '\n') && (i < size) )

            /*
             *  determine length of next word
             */

            i++;
        if ( (total + i) <= size )
            {

            /*
             *  enuf room on current line for this word, copy it
             */

            for ( j = 0; j < i; j++ )
                *p++ = *q++;
            total += i;
            if ( !(*q) )

            /*
             *  end of string
             */

                {
                *p = *q;
                return( strlen(s) );
                }
            else if ( *q == '\n' )
                {
```

```
            *p++ = ' ';
            *p = '\0';
            return( strlen(s) );
            }
        else if ( *q != ' ' )

        /*
         *  one long word
         */

            {
            *p++ = '\0';
            return( size );
            }

        /*
         *  find start of next word
         */

        while ( *q == ' ' )
            {
            *p++ = *q++;
            total++;
            }
        }
    else

    /*
     *  not enough room on line for next word, return line so far
     */

        {
        *p = '\0';
        return( total );
        }
    }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  x_offset()
*
*
* MODULE FUNCTION:
*
*   Determines the horizontal offset to center a string in a symbol.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

int  x_offset( w, str )

    int     w;
    char    *str;

{
    /*
     *  from the center of the width, back up 1/2 the width required for the
     *  text using the current font.
     */

    int i = ( (w/2) - (XTextWidth(fs, str, strlen(str))/2 ) );

    /*
     *  offset must be at least 3 pixels from the center.
     */

    return( (i > 3) ? i : 3 );
}
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    images.h
*
*
* FILE FUNCTION:
*
*   This file contains the global variables and function prototypes for images.c
*
*
* SPECIFICATION DOCUMENTS:
*
*   /home/project/3531/Docu/GCB.spec.doc
*
*
* FILE MODULES:
*
*   N/A
*
*****************************************<----->*****************************************/

/*
 *  Coordinates of rectangle that contains the drawings/symbols shouldn't collide with
 *  canvas borders, so move ulc in a little.
 */

#define xx 3
#define yy 3

/*
 *  Global variables used in images.c
 */

Drawable    drawable;

GContext    gcon;

XFontStruct *fs;

int         Red,
            insensitive_color;


/*
 *  Function prototypes for images.c
 */

char    *longest_line();

int     draw_text();

void    draw_begin(),
        draw_end(),
        draw_goto(),
        draw_if (),
        draw_pause(),
        draw_print(),
        draw_set(),
        draw_start(),
        draw_stop(),
        draw_text_symbol();
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    init_X.c
*
* FILE FUNCTION:
*
*   This file contains the routines which create and initialize the X Windows/MOTIF
*   interface of the GCB.
*
*
* FILE MODULES:
*
*   init_graphics() -   This routine initializes the X Windows connection and defines
*                       the X Windows widgets.
*
*****************************************<----->*****************************************/

#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/cursorfont.h>
#include <X11/Shell.h>
#include <X11/StringDefs.h>
#include <Xm/Xm.h>

#include <Xm/BulletinB.h>
#include <Xm/CascadeB.h>
#include <Xm/DialogS.h>
#include <Xm/DrawingA.h>
#include <Xm/Form.h>
#include <Xm/Frame.h>
#include <Xm/Label.h>
#include <Xm/MainW.h>
#include <Xm/MessageB.h>
#include <Xm/PushB.h>
#include <Xm/RowColumn.h>
#include <Xm/ScrolledW.h>
#include <Xm/ScrollBar.h>
#include <Xm/Text.h>
#include <Xm/ToggleB.h>

#include "gcb.h"
#include "pixmaps.h"
#include "next_inputs.h"
#include "widgets.h"
#include "lines.h"
#include "constants.h"
#include "cbr.h"
#include "init_X.h"
#include "fonts.h"
#include "cursors.h"
#include "gcb_icon.h"
#include "element_file.h"


void    get_colors();

char    help_str[] = "Select the button to display HELP for";
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  init_graphics()
*
* MODULE FUNCTION:
*
*   This routine builds the MOTIF-based user interface.  This routine builds all
*   the forms and popups which comprise the interface.  This routine also installs
*   the callbacks which control the interface.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  init_graphics()

{
    Arg                     args[10];
    int                     n;
    Pixel                   background,
                            foreground;
    Pixmap                  icon_pixmap,
                            pixmap;
    XImage                  *image;
    XSetWindowAttributes    attribs;
    Font                    sm_font, lg_font;
    GContext                gcon;
    XFontStruct             *fs;

    image = (XImage *) CreateDefaultImage( questionBits, 22, 22 );
    XmInstallImage( image, "question_img" );

    /*
     *  Create the main application shell and the main window.
     */

    top = XtAppCreateShell( "gcb","GCB", applicationShellWidgetClass, display,NULL,0);

    /*
     *  Are we on a color or a mono display?
     */

    if ( DisplayPlanes(display,DefaultScreen(display)) == 1 )
        Color = False;
    else
        Color = True;

    /*
     *  Create and install the GCB icon.
     */

    icon_pixmap = XCreateBitmapFromData( display, RootWindowOfScreen(XtScreen(top)),
                                         gcb_icon_bits, gcb_icon_width, gcb_icon_height);
    XtSetArg( args[0], XtNiconPixmap, icon_pixmap );
    XtSetValues( top, args, 1 );

    /*
     *  Create the cursors for line draw, print, and delete
     */
```

```
    tcross_cursor   = XCreateFontCursor( display, XC_crosshair);
    ul_cursor       = XCreateFontCursor( display, XC_ul_angle );
    lr_cursor       = XCreateFontCursor( display, XC_lr_angle );
    basic_cursor    = XCreateFontCursor( display, XC_top_left_arrow);
    clock_cursor    = XCreateFontCursor( display, XC_watch);


    win_main = XmCreateMainWindow( top, "win_main", NULL, 0 );
    XtManageChild( win_main );


    /*
     *  Create the main menu bar and its associated menu pane which is where
     *  the menu buttons/cascades are installed.
     */

    n = 0;
    XtSetArg( args[n], XmNresizeWidth, False); n++;
    XtSetArg( args[n], XmNresizeHeight, False); n++;
    XtSetArg( args[n], XmNwidth, 1000); n++;
    XtSetArg( args[n], XmNheight, 30); n++;
    mnb_main = XmCreateMenuBar( win_main, "mnb_main", args, n );
    XtManageChild( mnb_main );
whirl();

    /*
     *  Create "Position" cascade menu items and menu.
     */

    menu_pane = cr_pulldown( NULLS, mnb_main );

    cr_command( NULLS, menu_pane, "Select Position", cbr_sel_pos, 0 );
    cr_command( NULLS, menu_pane, "Create Position", cbr_cre_pos, 0 );
    cr_command( NULLS, menu_pane, "Exit GCB",        cbr_exit, 0 );

    cascade   = cr_cascade ( NULLS, mnb_main, menu_pane, NULL, "Position" );
whirl();

    /*
     *  Create "Comp" cascade menu items and menu.
     */

    menu_pane = cr_pulldown( NULLS, mnb_main );

    cr_command ( NULLS, menu_pane, "Select   Comp", cbr_sel_comp, MANAGE );
    cr_command ( NULLS, menu_pane, "Create   Comp", cbr_cre_comp, 0);
    menu_pullrite = cr_pulldown( NULLS, menu_pane );

    cr_cascade( NULLS,menu_pane, menu_pullrite, NULL, "Edit     Comp Attributes");
    cr_command( NULLS,menu_pullrite,"Comp Purpose",          cbr_purpose,   0);
    cr_command( NULLS,menu_pullrite,"Select Root Element",   cbr_root_elem, MANAGE);

whirl();
    menu_pullrite = cr_pulldown( NULLS, menu_pane );
    cr_cascade( NULLS, menu_pane, menu_pullrite, NULL, "Print    Comp" );
    cr_command( NULLS, menu_pullrite,"Print RMS  Report",cbr_print, PRINT_REP1 );
    cr_command( NULLS, menu_pullrite,"Print FIDO Report",cbr_print, PRINT_REP1 );
    cr_command( NULLS, menu_pullrite,"Print INCO Report",cbr_print, PRINT_REP1 );

    cr_command( NULLS,menu_pane, "Display  Comp Callflow", cbr_call_flow, 0);
    cr_command( NULLS,menu_pane, "Install  Comp",          cbr_build,     BUILD_COMP);
    cr_command( NULLS,menu_pane, "Validate Comp",          cbr_validate,  0);
whirl();

    cascade   = cr_cascade( NULLS, mnb_main, menu_pane, NULL, "Comp");

    /*
     *  Create "Element" cascade menu items and menu.
     */

    menu_pane = cr_pulldown( NULLS, mnb_main );

    cr_command( NULLS, menu_pane, "Select  Element", cbr_elem_popup, 3);
    cr_command( NULLS, menu_pane, "Create  Element", cbr_elem_popup, 0);
    cr_command( NULLS, menu_pane, "Edit    Element Purpose", cbr_purpose, 1 );
    cr_command( NULLS, menu_pane, "Delete  Element", cbr_elem_popup, 2 );
    cr_command( NULLS, menu_pane, "Print   Element", cbr_print,      PRINT_REP  );
whirl();

    cr_command( NULLS, menu_pane,    "Save    Element", cbr_sav_element,0 );
    cr_command( NULLS, menu_pane,    "Copy    Element", cbr_elem_popup, 1 );

    menu_pullrite  = cr_pulldown( NULLS, menu_pane );
    cr_cascade( NULLS, menu_pane,menu_pullrite,NULL,"Audit   Element");

    menu_pullrite2  = cr_pulldown(NULLS, menu_pullrite );
    cr_command( NULLS, menu_pullrite2,"Check Lines",           cbr_audit, JUST_LINES );
    cr_command( NULLS, menu_pullrite2,"Check Expressions",     cbr_audit, JUST_EXPR );
    cr_command( NULLS, menu_pullrite2,"Lines and Expressions",cbr_audit, LINES_AND_EXPR);
    cr_cascade( NULLS, menu_pullrite, menu_pullrite2, NULL,"Show  Audit");
    cr_command( NULLS, menu_pullrite, "Clear Audit",           cbr_audit, 4 );
    cr_command( NULLS, menu_pane,     "Install Element",       cbr_build, BUILD_ELEMENT);

    cascade = cr_cascade(NULLS, mnb_main, menu_pane, NULL, "Element");

whirl();

    /*
     *  Create "Macro" cascade menu items and menu.
     */

/*DEBUG
    menu_pane = cr_pulldown( NULLS, mnb_main );

    cr_command( NULLS, menu_pane, "Begin Macro Capture", cbr_exit, 0);
    cr_command( NULLS, menu_pane, "End Macro Capture",   cbr_exit, 0);
    cr_command( NULLS, menu_pane, "Execute Macro",       cbr_exit, 0);

    cascade   = cr_cascade ( NULLS, mnb_main, menu_pane, NULL, "Macro");
    whirl();
*/

    /*
     *  Create "Options" cascade menu items and menu.
     */

    menu_pane = cr_pulldown( NULLS, mnb_main );

    cr_command( NULLS, menu_pane, "Show Option Settings", cbr_show_status, MANAGE);

    menu_pullrite   = cr_pulldown(NULLS, menu_pane );
    cr_cascade( NULLS, menu_pane, menu_pullrite, NULL, "Set Symbol Display");
    cr_command( NULLS, menu_pullrite, "Show Logical Text",  cbr_set_sym_display, 1);
    cr_command( NULLS, menu_pullrite, "Show Expression Text", cbr_set_sym_display, 2);

    menu_pullrite = cr_pulldown( NULLS, menu_pane );
```

```
    cr_cascade( NULLS, menu_pane, menu_pullrite, NULL, "Set Symbol Snap");
    cr_command( NULLS, menu_pullrite, "Symbol Snap On",  cbr_snap, 1);
    cr_command( NULLS, menu_pullrite, "Symbol Snap Off", cbr_snap, 2);

    menu_pullrite   = cr_pulldown(NULLS, menu_pane );
    cr_cascade( NULLS, menu_pane, menu_pullrite, NULL, "Set Audit");
    cr_command( NULLS, menu_pullrite, "Audit On",  cbr_audit_on, 1);
    cr_command( NULLS, menu_pullrite, "Audit Off", cbr_audit_on, 2);

    cr_command( NULLS, menu_pane, "Set Colors",cbr_color_menu, 0);
    cr_command( NULLS, menu_pane, "Set Target Language", cbr_language_menu,0);


    cascade = cr_cascade( NULLS, mnb_main, menu_pane, NULL, "Options");

whirl();

    /*
     *  Create "Help" button in main menubar.
     */

    menu_pane = cr_pulldown( NULLS, mnb_main );
    cr_command( NULLS, menu_pane, "Work Area",      cbr_help, WORK_AREA );
    cr_command( NULLS, menu_pane, "Palette Area",   cbr_help, PALETTE_AREA );
    cr_command( NULLS, menu_pane, "Browse Manual", cbr_help, BROWSE );

    cascade = cr_cascade( NULLS, mnb_main, menu_pane, NULL, "Help");

    XtSetArg( args[0], XmNmenuHelpWidget, (XtArgVal) cascade);
    XtSetValues( mnb_main, args, 1 );
whirl();

    /*
     *  Create container form
     */

    frm_container = cr_form( NULLS, win_main, NULL, 0 );
whirl();

    /*
     *  Create the "Set Symbol Attributes" popup.
     */

    build_log_attribs( frm_container );

    /*
     *  create help popup
     */

    build_help( frm_container );
whirl();

    /*
     *  create call popup
     */

    build_call_popup( frm_container );
    build_start_popup( frm_container );

whirl();
    /*
     *  create pause popup
     */

    build_pause_popup( frm_container );
whirl();


    /*
     *  create print/text popup
     */

    build_text_popup( frm_container );
whirl();

    /*
     *  create element selection popup
     */

    build_sel_elem_popup( frm_container );
whirl();

    /*
     *  create select root element popup
     */

    build_sel_root_elem( frm_container );
whirl();

    /*
     *  create element creation popup
     */

    build_cre_elem_popup( frm_container );
whirl();

    /*
     *  create element copy popup
     */

    build_copy_elem_popup( frm_container );
whirl();

    /*
     *  create element deletion popup
     */

    build_del_elem_popup( frm_container );
whirl();

    /*
     *  create comp selection popup
     */

    build_sel_comp_popup( frm_container );
whirl();

    /*
     *  create comp creation popup
     */

    build_cre_comp_popup( frm_container );
whirl();

    /*
     *  create print element popup
     */
```

```
    build_print_elem_popup( frm_container );
whirl();

    /*
     *  create purpose popup for both comp and element.
     */

    build_purpose_popup( frm_container );
whirl();

    /*
     *  create create position popup
     */

    build_cre_pos_popup( frm_container );
whirl();

    /*
     *  create select position popup
     */

    build_sel_pos_popup( frm_container );
whirl();

    /*
     *  create select target language popup
     */

    build_lan_select_popup( frm_container );
whirl();

    /*
     *  Build the Show Status popup.
     */

    build_status_popup( frm_container );
whirl();

    /*
     *  Build the Displayer which is used to display messages during source
     *  code generation and compiling.
     */

    build_displayer_popup( frm_container );

    /*
     *  Create status frame
     */

    frame_status  = cr_frame( NULLS, frm_container, NULL, NULL );

    /*
     *  Create form to hold status items
     */

    frm_status = cr_form( NULLS, frame_status, NULL, NULL );
    set_attribs( FORM, frm_status, 278, 270, XmRESIZE_NONE );


    /*
     *  Create status label widgets
     */

    cr_label( NULLS, frm_status,"Position ", 0, IGNORE, IGNORE, IGNORE, IGNORE);
    txt_position = cr_label( NULLS, frm_status, "INCO", 0, IGNORE,IGNORE,40,90);

    cr_label( NULLS, frm_status, "Comp ", 0, 8, IGNORE , IGNORE, IGNORE);
    txt_comp     = cr_label( NULLS, frm_status, "comp", 0, 8, IGNORE, 40, 90);

    cr_label( NULLS, frm_status, "Element ",  0, 16, IGNORE,IGNORE, IGNORE);
    txt_element  = cr_label( NULLS, frm_status, "ELEMENT", 0, 16,IGNORE,40,90);

    cr_label( NULLS, frm_status, "Author ",   0, 24, IGNORE,IGNORE, IGNORE);
    txt_author   = cr_label( NULLS, frm_status, "AUTHOR", 0, 24, IGNORE, 40,90);

    cr_label( NULLS, frm_status, "Created ",  0, 32, IGNORE, IGNORE,IGNORE);
    txt_created  = cr_label( NULLS, frm_status, " ", 0, 32, IGNORE, 40,90);

    cr_label( NULLS, frm_status, "Last Update ", 0, 40, IGNORE, IGNORE, IGNORE);
    txt_last_update = cr_label( NULLS, frm_status, " ", 0, 40,IGNORE,40,90);

    XtSetArg( args[0], XmNalignment,  XmALIGNMENT_BEGINNING);
    XtSetValues( txt_position, args, 1 );
    XtSetValues( txt_comp, args, 1 );
    XtSetValues( txt_element, args, 1 );
    XtSetValues( txt_author, args, 1 );
    XtSetValues( txt_created, args, 1 );
    XtSetValues( txt_last_update, args, 1 );
whirl();

    /*
     *  Create the Comp/Element purpose toggle buttons.
     */

    cr_label( NULLS, frm_status, "Purpose", 0, 54, IGNORE, IGNORE,IGNORE);

    rb_purpose = cr_radio_box( NULLS, frm_status, XmHORIZONTAL );
    set_position(rb_purpose, 52, IGNORE, 37, IGNORE);

    tgl_comp    = Ncr_toggle( NULLS, rb_purpose, "Comp",    cbr_tgl_purpose, 2, 2 );
    tgl_element = Ncr_toggle( NULLS, rb_purpose, "Element", cbr_tgl_purpose, 3, 3 );
    arm_tgl( tgl_element );

    txt_purpose    = cr_scr_text( NULLS, frm_status, 5, False, 260, 10, 165 );
    XtSetArg( args[0], XmNsensitive, True);
    XtSetArg( args[1], XmNwordWrap,  True);
    XtSetValues( txt_purpose, args, 2);

    /*
     *  create Frame to hold cancel form
     */

    frame_cancel = cr_frame( NULLS, frm_container, NULL, frame_status);

    frm_cancel   = cr_form ( NULLS, frame_cancel,  NULL, NULL );
    set_attribs( FORM, frm_cancel, 278, 70, XmRESIZE_NONE );

    /*
     *  create labels for mode and status
     */

    cr_label( NULLS, frm_cancel, "MODE: ",  0, 17,IGNORE, 5, IGNORE);
    txt_mode   = cr_label(NULLS, frm_cancel, "Edit Symbol", 0, 17, IGNORE, 30, IGNORE);
    cr_label( NULLS, frm_cancel, "STATUS: ", 0,52,IGNORE, 5, IGNORE );
    txt_status = cr_label(NULLS, frm_cancel, "Incomplete", 0, 52, IGNORE, 30, IGNORE);

    /*
     *  create cancel button widget
```

```
     */

    btn_cancel = cr_command( NULLS, frm_cancel, "CANCEL", cbr_cancel, 0 );
    set_position( btn_cancel, 15, IGNORE, 74, IGNORE );

    whirl();

    /*
     *  setup palette
     */

    get_colors();
    setup_palette( frm_container, frame_cancel );

    whirl();


    /*
     *  setup colors popup
     */

    if (Color)
        build_color_popup( frm_container );

    whirl();

    /*
     *  setup math menu - overwrites palette for if/set statement
     */

    setup_math_menu(frm_container, frame_cancel);
    whirl();

    /*
     *  Setup the matrix menu - overwrites the math menu.
     */

    setup_matrix_menu( frm_container, frame_cancel );
    whirl();

    /*
     *  Setup the trigometric menu - overwrites the math menu.
     */

    setup_trig_menu( frm_container, frame_cancel );
    whirl();

    /*
     *  create logic expr entry popup
     */

    build_log_popup( frm_container );
whirl();

     /*
      *  Build ask() popup dialog.
      */

    dlg_ask  = cr_popup( NULLS, top, "Ask" );
    frm_ask  = cr_form ( NULLS, dlg_ask, NULL, NULL );

    XtSetArg( args[0], XmNforeground, &foreground );
    XtSetArg( args[1], XmNbackground, &background );
    XtGetValues( dlg_ask, args, 2 );

    pixmap  = XmGetPixmap( XtScreen(dlg_ask), "question_img", foreground, background );
    cr_pixmap( NULLS, frm_ask, &pixmap, 0, IGNORE, 1, IGNORE );

    lbl_ask = cr_label( NULLS, frm_ask, NULLS,  0, 0, IGNORE, 0, 100 );

    cr_separator( NULLS, frm_ask, 40, 45, 0, 100 );

    btn_ask_y = Ncr_rel_cmd( NULLS, frm_ask, "YES",  cbr_clear_popup, ASK_YES, 65, 10);
    btn_ask_n = Ncr_rel_cmd( NULLS, frm_ask, "NO",   cbr_clear_popup, ASK_NO,  65, 41);
    btn_ask_h = Ncr_rel_cmd( NULLS, frm_ask, "Help", cbr_help,        ASK,     65, 75);
whirl();

    /*
     *  Build user_ack() popup dialog.
     */

    dlg_ack = cr_popup( NULLS, top, "Acknowledgement" );
    frm_ack = cr_form ( NULLS, dlg_ack, NULL, NULL );
    set_attribs( FORM, frm_ack, 100, 70, XmRESIZE_NONE );

    image = (XImage *) CreateDefaultImage(infoBits, 11, 24);
    XmInstallImage(image, "info_img");

    XtSetArg( args[0], XmNforeground, &foreground );
    XtSetArg( args[1], XmNbackground, &background );
    XtGetValues( dlg_ack, args, 2 );

    pixmap = XmGetPixmap( XtScreen(dlg_ack), "info_img", foreground, background );
    cr_pixmap( NULLS, frm_ack, &pixmap, 0, IGNORE, 1, IGNORE );

    lbl_ack = cr_label    ( NULLS, frm_ack, NULLS,  0, 0, IGNORE, 0, 100 );

    cr_separator( NULLS, frm_ack, 40, 45, 0, 100 );

    Ncr_rel_cmd( NULLS, frm_ack,"OK",   cbr_clear_popup,USER_ACK, 61, 15);
    Ncr_rel_cmd( NULLS, frm_ack,"Help", cbr_help,  USER_ACK_HELP, 63, 70);
whirl();

    /*
     *  create h and v ruler bars.
     */

    n = 0;
    XtSetArg( args[n], XmNwidth,        16  ); n++;
    XtSetArg( args[n], XmNheight,       800 ); n++;
    XtSetArg( args[n], XmNresizePolicy, XmRESIZE_NONE ); n++;
    v_scroll = XmCreateDrawingArea(frm_container, "v_scroll", args, n);
    XtManageChild( v_scroll );
    set_attach_widget(v_scroll, mnb_main, NULL, frame_status, NULL);

    n = 0;
    XtSetArg( args[n], XmNwidth,        800 ); n++;
    XtSetArg( args[n], XmNheight,       16  ); n++;
    XtSetArg( args[n], XmNresizePolicy, XmRESIZE_NONE ); n++;
    h_scroll = XmCreateDrawingArea(frm_container, "h_scroll", args, n);
    XtManageChild( h_scroll );
    set_attach_widget(h_scroll, mnb_main, NULL, v_scroll, NULL);


    XtAddEventHandler( v_scroll, ExposureMask, FALSE, cbr_scroll_expose, NULL);
    XtAddEventHandler( h_scroll, ExposureMask, FALSE, cbr_scroll_expose, NULL);
whirl();
```

```
    /*
     *  create h and v sliding rules .
     */
    n = 0;
    XtSetArg( args[n], XmNwidth,     13 ); n++;
    XtSetArg( args[n], XmNheight,    1  ); n++;
    XtSetArg( args[n], XmNy,         5  ); n++;
    XtSetArg( args[n], XmNx,         -4 ); n++;
    h_rule = XmCreateDrawingArea(v_scroll, "h_rule", args, n);
    XtManageChild( h_rule );

    n = 0;
    XtSetArg( args[n], XmNwidth,     1  ); n++;
    XtSetArg( args[n], XmNheight,    13 ); n++;
    XtSetArg( args[n], XmNx,         5  ); n++;
    XtSetArg( args[n], XmNy,         -4 ); n++;
    v_rule = XmCreateDrawingArea(h_scroll, "v_rule", args, n);
    XtManageChild( v_rule );

    XtAddEventHandler( h_rule, ExposureMask, FALSE, cbr_rule_expose, NULL);
    XtAddEventHandler( v_rule, ExposureMask, FALSE, cbr_rule_expose, NULL);
whirl();

    /*
     *  create Scrolled Window to hold drawing area widget
     */

    n = 0;
    XtSetArg( args[n], XmNwidth,            800 ); n++;
    XtSetArg( args[n], XmNheight,           800 ); n++;
    XtSetArg( args[n], XmNscrollingPolicy,  XmAUTOMATIC ); n++;
    scr_WA = (Widget)XmCreateScrolledWindow(frm_container, NULLS, args, n);
    XtManageChild( scr_WA );
    set_attach_widget(scr_WA, h_scroll, NULL, v_scroll, NULL);

    /*
     *  create drawing area widget
     */

    n = 0;
    XtSetArg( args[n], XmNwidth,  MAIN_CANVAS_WIDTH+MAIN_CANVAS_WIDTH/2 - 350);  n++;
    XtSetArg( args[n], XmNheight, MAIN_CANVAS_HEIGHT+MAIN_CANVAS_HEIGHT/2-50); n++;
    XtSetArg( args[n], XmNresizePolicy, XmRESIZE_NONE ); n++;
    XtSetArg( args[n], XmNinputCallback,WA_input_code ); n++;
    draw_area = XmCreateDrawingArea( scr_WA, NULLS, args, n );
    XtManageChild( draw_area );

    XtSetArg( args[0], XmNworkWindow, draw_area );
    XtSetValues( scr_WA, args, 1 );

    XtSetArg( args[0], XtNbackground, &LDbackground );
    XtSetArg( args[1], XtNforeground, &LDforeground );
    XtGetValues( draw_area, args, 2 );

    XtAddEventHandler( draw_area, ExposureMask, FALSE, cbr_expose, (Opaque)NULL );
    XtAddEventHandler( draw_area, EnterWindowMask , FALSE, cbr_enter_canvas,
        (Opaque)NULL );
    XtAddEventHandler( draw_area, PointerMotionMask | ButtonMotionMask,
        FALSE, cbr_motion_canvas, (Opaque)NULL );

whirl();

    /*
     *  create popup menus in drawing area
     */

    bboard = cr_popup( NULLS, draw_area, NULLS );

    Lmenu_popup = XmCreatePopupMenu( bboard, NULLS, NULL, 0);
    menu_list[0] = cr_command( NULLS, Lmenu_popup, "Move",     cbr_set_anchor, MoveBox );
    menu_list[1] = cr_command( NULLS, Lmenu_popup, "Copy",     cbr_set_anchor, CopyBox );
    menu_list[2] = cr_command( NULLS, Lmenu_popup, "Delete",   cbr_set_anchor, DeleteBox);
    menu_list[3] = cr_command( NULLS, Lmenu_popup, "Zoom Out",menu_proc,       3 );
    menu_list[4] = cr_command( NULLS, Lmenu_popup, "Print",    cbr_set_anchor, PrintBox );
    menu_list[5] = cr_command( NULLS, Lmenu_popup, "Undo",     cbr_cancel,     5 );
    menu_list[6] = cr_command( NULLS, Lmenu_popup, "Exit",     cbr_exit,       6 );

    Rmenu_popup = XmCreatePopupMenu( bboard, NULLS, NULL, 0 );
    cr_command( NULLS, Rmenu_popup, "Select Comp",     cbr_sel_comp,     MANAGE );
    cr_command( NULLS, Rmenu_popup, "Select Element", cbr_elem_popup,  3 );
    cr_command( NULLS, Rmenu_popup, "Save   Element", cbr_sav_element, 0 );

    menu_pullrite   = cr_pulldown(NULLS, Rmenu_popup );
    cr_cascade( NULLS, Rmenu_popup, menu_pullrite, NULL, "Set Symbol Display");
    cr_command( NULLS, menu_pullrite, "Show Logical Text",    cbr_set_sym_display, 1);
    cr_command( NULLS, menu_pullrite, "Show Expression Text", cbr_set_sym_display, 2);

    menu_pullrite = cr_pulldown( NULLS, Rmenu_popup );
    menu_item = cr_cascade( NULLS, Rmenu_popup, menu_pullrite, NULL, "Audit  Element");
    menu_pullrite2 = cr_pulldown( NULLS, menu_pullrite );
    cr_command( NULLS, menu_pullrite2,"Check Lines",           cbr_audit, JUST_LINES);
    cr_command( NULLS, menu_pullrite2,"Check Expressions",     cbr_audit, JUST_EXPR);
    cr_command( NULLS, menu_pullrite2,"Lines and Expressions",cbr_audit, LINES_AND_EXPR);
    cr_cascade( NULLS, menu_pullrite, menu_pullrite2, NULL,"Show Audit");

    cr_command( NULLS, menu_pullrite, "Clear Audit",           cbr_audit, 4);

    XtSetArg( args[0], XmNwhichButton, Button1 );
    XtSetValues(Lmenu_popup, args, 1);

whirl();

    /*
     *  create line draw gc using a symbol canvas as a model
     */

    LDgc = XCreateGC(display,RootWindow(display,DefaultScreen(display)), 0, NULL);
    XSetForeground( display, LDgc, BlackPixel(display, DefaultScreen(display)) ^
        LDbackground);
    XSetFunction(display, LDgc, GXxor);

    /*
     *  create work area gc using a symbol canvas as a model
     */

    WAgc = XCreateGC(display,RootWindow(display,DefaultScreen(display)),0,NULL);

    /*
     *  give wagc the default fore and backgrounds of a palette item
     */

    XSetForeground(display, WAgc, BlackPixel(display, DefaultScreen(display)));
    XSetBackground(display, WAgc, WhitePixel(display, DefaultScreen(display)));
    XSetGraphicsExposures(display, WAgc, False);

    XtRealizeWidget( top );
    XtMapWidget( frame_palette );
```

```
    attribs.backing_store = Always;
    XChangeWindowAttributes(display, XtWindow(draw_area), CWBackingStore, &attribs);

    Mode = EditSymbol;

    big_font = XLoadFont(display, "9x15");
    small_font = XLoadFont(display, "fixed");
    teeny_font = XLoadFont(display, "6x10");
    XSetFont(display, WAgc, WAFont = (Font)big_font);

whirl();

}
```

```
/*************************************<----->*************************************
*
* FILE NAME:    init_X.h
*
* FILE FUNCTION:
*
*   Variable declarations, constants, and function prototypes for the init_X.c
*   routines.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /home/project/3531/Docu/GCB.spec.doc
*
*
* FILE MODULES:
*
*   N/A
*
*************************************<----->*************************************/

#define PANEL_WIDTH         275
#define PANEL_HEIGHT        470
#define SUBCANVAS_WIDTH     500
#define SUBCANVAS_HEIGHT    500

#define LINECURSOR          1
#define DELETECURSOR        2


int PopupStat[5],
    Zoomed;

char *creat_str[] =
        {   "Select the type of comp to create.  A comp is contained in a its own",
            "directory and is composed of comp elements and library elements.  A ",
        };

char *read_str[] =
        {   "Enter the name of the comp element or library element file to read.",
            "The filename may either be typed in or selected from the Comp List ",
            "using the mouse.                                                   ",
        };

/*
 *  Callback lists used in the building of the MOTIF interface.
 */

XtCallbackRec   WA_input_code[] = {
    { (XtCallbackProc) cbr_canvas,   (caddr_t) NULL },
    { (XtCallbackProc) NULL,         (caddr_t) NULL }
    };

XtCallbackRec   symbol_input_code[] = {
    { (XtCallbackProc) cbr_symbol,   (caddr_t) NULL },
    { (XtCallbackProc) NULL,         (caddr_t) NULL }
    };
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  init_vars()
*
* MODULE FUNCTION:
*
*   This routine initializes variables like the logfile name and target language.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.01 - 08/01/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

#include <stdio.h>
#include <pwd.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include "gcb.h"


int  init_vars( argc, argv )

    int     argc;
    char    *argv[];

{
    struct passwd   *user;
    int             rc;


    /*
     *  Default the logfile and several other files to the local directory for now.
     *  The user may change their location in the defaults file later.
     */

    ErrorLogLevel = 1;      /* 1=normal, 3=debug */

    strcpy( LogFile,        "./LogFile.GCB"   );
    strcpy( DisplayFile,    "./DisplayOutput" );
    strcpy( MSIDTable,      "./MSIDTable.GCB" );
    strcpy( UserFuncsPath,  "./" );
    strcpy( WSGlobals,      "./WSGlobals.GCB" );

    /*
     *  Default the target language to MOAL for now, the user may specify a new
     *  value in the defaults file.
     */

    TargetLanguage  = MOAL;

    /*
     *  Initialize the cell map, the symbol array, and the element/comp file
     *  variables.
     */

    init_element_vars();
    init_comp_vars();

    PositionPath[0] = '\0';
    CompDir[0]      = '\0';


    /*
     *  If there is a PositionPath, get the current directory, we will
     *  return here on exit.
     */

    load_curr_dir( Swd );

    /*
     *  Determine who the user is.  Get the current user's id,
     *  then locate the user's id in the password file.
     */

    if ((user = getpwuid( getuid() )) == NULL )
        {
        elog(1,"init_vars() - couldn't get user name");
        UserName[0] = NULL;
        }
    else
        strcpy( UserName, user->pw_gecos );

    /*
     *  See if a defaults file exists in the local directory.  If a defaults
     *  file is not found locally, try the user's home directory.
     */

    strcpy( DefaultsFile, "./.Defaults.GCB" );
    if ((rc = open_read_defaults()) < 0 )
        {
        if ( rc == NOT_FOUND )
            {

            /*
             *  Try to find defaults file in home directory.
             */

            strcpy( DefaultsFile, user->pw_dir );
            strcat( DefaultsFile, "/.Defaults.GCB" );
            rc = open_read_defaults();
            }
        }

    /*
     *  No defaults file was found.  Make sure defaults
     *  are initialized to a safe state.  Leave the defaults
     *  file path set to the user's home directory, that is
     *  where the new defaults file will be created on exit.
     */

    if ( rc )
        {
        elog(1,"init_vars() - no defaults file found");

        GElementFile[0] = NULL;
        GCompFile[0]    = NULL;
        strcpy( Author,      UserName );
        strcpy( MacrosPath,  "./" );
        strcpy( MSIDTable,   "./MSIDTable.GCB" );
        strcpy( WSGlobals,   "./WSGlobals.GCB" );
        strcpy( DisplayFile, "./DisplayOutput" );
        }
    else
        if ( PositionPath[0] )
            if ( chdir(PositionPath) )
```

```
            ValidPosition = FALSE;
        else
            {
            ValidPosition = TRUE;
            if ( CompDir[0] )
                if ( chdir(CompDir) )
                    ValidComp = FALSE;
                else
                    ValidComp = TRUE;
            }

    /*
     *  See if the command line contains a file name.
     */

    if ( argc > 1 )
        {
        strcpy( ElementFile,  argv[1] );
        strcpy( GElementFile, ElementFile );
        strcat( GElementFile, EL_EXT  );
        }

    /*
     *  See if the command line contained a new Error Log Level.
     */

    if ( argc > 2 )
        ErrorLogLevel = atoi( argv[2] );

    /*
     *  See if we have a filename, if so, try to read it in.  If there is an error
     *  reading the file, let the user start a new one.
     */

    if ( ValidComp )
        read_comp_file();

    if ( GElementFile[0] )
        if ( read_element_file() )
            init_element_vars();

    /*
     *  Update the position panel text strings.
     */

    upd_pos_panel( NO_CHANGE );

    /*
     *  Update the mode indicators.
     */

    upd_mode_panel();
    return( OK );
}
```

```
%{
#include "y.tab.h"
%}
RO          [=|<|>|<=|>=|<>]
NUM         [0-9]
F           FN_
WSL         [WS_][a-zA-Z0-9_]+
OBL         [V][a-zA-Z0-9_]+
L           [a-zA-Z][a-zA-Z0-9_]*
S           [\"][ -~]*[\"]
AO          [or|xor]
SIGN        [+|-]
PER         \.
%%
[ \t]       ;
[(]         { return( L_PERIN ); }
[)]         { return( R_PERIN ); }
[\[]        { return( L_BRACK ); }
[\]]        { return( R_BRACK ); }

">>" |
"<<"        { return( SHIFT ); }

">=" |
"<=" |
"<>" |
">"  |
"<"         {  return( REL_OPER ); }


":="        { return( SET_EQ ); }

"="         { return( EQEQ ); }

"cos" |
"acos" |
"sin" |
"asin" |
"tan" |
"atan" |
"cuber" |
"ln" |
"cosh" |
"exp" |
"sinh" |
"tanh" |
"sqrt" |
"log" |
"nlog" |
"activate" |
"terminate" { return( FIXED_FUNC ); }

"power"     { return( POWER ); }

"PI"        { return ( PI ); }

"not"       { return( NOT ); }

"*"         { return( MUL ); }

"/"         { return( DIV ); }

"+"         { return( ADD ); }

"-"         { return( MINUS ); }

"ADD"       { return( MADD ); }

"SUB"       { return( MSUB ); }

"MULT"      { return( MMULT ); }

"IDENT"     { return( IDENT ); }

"INVERSE"   { return( INVERS ); }

"TRANSP"    { return( TRANSP ); }

"CROSS"     { return( CROSS ); }

"DOT"       { return( DOT ); }

"and" |
"or"        { return( LOG_OPER ); }


"bitAnd"    |
"bitOr"     |
"bitXor"    { return( BIT_OPER ); }


"-"         { return( SIGN ); }

{S}         { return( STRING ); }

{NUM}*"."{NUM}*     { return( UNSIGNED_REAL ); }
"0x"{NUM}|
"0X"{NUM}           { return( UNSIGNED_HEX ); }
{NUM}+      { return( UNSIGNED_INT ); }

{PER}       { return( PERIOD ); }

","         { return( COMMA ); }

{F}{L}      { return(USER_FUNC); }

{WSL}       { process_id(); return( WSID ); }

{OBL}       { process_id(); return( OBID ); }

{L}         { process_id(); return( ID ); }


%%

/*
 *  COTS include Files.
 */

#include <stdio.h>

/*
 * Custom include files.
 */

#include "symbol.h"
#include "gcb_parse.h"
```

```
/*
 *  Local defines.
 */

#define MAX_ERROR           1024

/*
 *  Global variables.
 */

char        *parse_data_ptr;
int         parse_error_value;
char        parse_error_message[ MAX_ERROR ];
extern char ElementFile[];

/*
 *  Function process_id is invoked when an indentifier is found in the token string, the
 *  function will determine if the identifier exists in the symbol table, if it does not
 *  exist, the appropriate error flags will be set and messages generated.
 */


process_id()
{
    char    local_error[ MAX_ERROR ];
    int     local_var = 1;

    /*
     *  If the symbol does not exist in the symbol table, note the error but continue
     *  parsing the expression.
     */

    if ( (yytext[0] == 'G') && (yytext[1] == 'V') && (yytext[2] == '_') )
        local_var = 0;

    if ( lookup_symbol( local_var ? ElementFile : NULL, yytext ) == NULL ) {
        sprintf( local_error, "Undeclared indentifier: %s\n", yytext );
        if ( (strlen( parse_error_message ) + strlen( local_error ) + 2) < MAX_ERROR )
            strcat( parse_error_message, local_error );
        parse_error_value |= UNDECLARED;
        }

    /*
     *  Process the data type of the ID being processed.
     */

    process_id_data_type( yytext );
}  /*  of function  */

/*
 *  The function input is used by the lexical analyzer to fetch a single byte of data
 *  from the input line.  The function is included to override the standard function
 *  provided by LEX.
 */

#undef input

int input()
{
    int     ch;

    ch = (int)*parse_data_ptr;
    if ( (ch != '\0') && (ch != ' ') )
        if ( stable_state )
            stable_state = 0;
    parse_data_ptr++;
    return( ch );
} /*  of function  */

/*
 *  The function unput is used by the lexical analyzer to replace the previous character
 *  on the unput data (essentialy a ungetc with a replace).  The function is included to
 *  overried the standard function provided by LEX.
 */

#undef unput

int unput( ch )
int ch;
{
    parse_data_ptr--;
    *parse_data_ptr = (char)ch;
}  /*  of function  */

/*
 *  The function output is used by the lexical analyzer to place a single character on
 *  the programs output stream.  The characters are simply thrown away because all
 *  diagnostics are presented using X-windowns.  The function is included to over-ride
 *  the standard function provided by LEX.
 */

#undef output

int output( c )
int c;
{
}  /*  of function  */
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    lines.c
*
*
* FILE FUNCTION:
*
*   This file contains most all of the routines to create, delete, and render
*   lines.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /home/project/3531/Docu/GCB.spec.doc
*
*
* FILE MODULES:
*
*   add_from_line()          - adds new line to the Symbol_Map structure for destination
*   cancel_line()            - erases line drawn, clears line field in start symbol
*   clear_cell_map_line()    - clears the cells occupied by the parameter line segment
*   clear_line()             - toggles the pixels in a line and so erases the line
*   delete_line()            - erases line, clears line fields in start and dest symbol
*   delete_lines()           - deletes all lines entering or leaving a symbol
*   draw_arrows()            - draws arrow at end of a line with the correct orientation
*   draw_line()              - sets the pixels in a line and so draws the line
*   draw_whole_line()        - draws all the segments of a line in either red or black
*   end_line()               - completes the line structures in the start and end symbols
*   erase_line()             - clears the pixels in the specified line
*   find_line_cell()         - finds the first cell in a line that is of type LINE_CELL
*   free_cellmap_linelists()- This routine frees all the linelists in all of the cells
*   free_lines()             - This routine deallocates all the space for all the lines
*   free_list()              - Routine frees all the lines in the parameter line list
*   free_segment()           - This routine frees the space in all the segments of a line
*   get_good_cell_pt()       - sets endpoint of a seg to last cell outside destination
*   mid_line()               - allocates new seg pointer when an 'elbow' is set in a line
*   prompt_true_false()      - determines if the user wants to draw a true or false line
*   set_cell_map_line()      - creates LineList structures to record cells in a line
*   set_line()               - sets the pixels in the specified line using GXcopy
*   start_line()             - creates new structures to start line; inits start sym
*   still_on_line()          - determines if the given point is on the given segment
*   valid_line()             - determines if the drawn line segment passes through a sym
*   zap_line()               - clears the pixels in a line and so erases the line
*
*****************************************<----->*****************************************/

#include <stdio.h>

#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xm/Xm.h>

#include "gcb.h"
#include "lines.h"
#include "widgets.h"
#include "element_file.h"
#include "images.h"


char tf_string[] = "Do you want to connect the TRUE or the FALSE portion?";
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  add_from_line()
*
*
* MODULE FUNCTION: This routine adds the new line to the Symbol_Map structure for
*   the destination symbol.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

void  add_from_line()
{
    LineList    *listPtr;


    /*
     *  See if there is already a list of lines which enter this symbol.
     */

    if ( LDendPtr->from == NULL )

        /*
         *  Need to malloc a new LineList struct.
         */

        {
        LDendPtr->from =  listPtr = (LineList *) malloc( sizeof(LineList) );
        listPtr->next = NULL;
        listPtr->prev = NULL;
        listPtr->line = LDlinePtr;
        listPtr->key  = LineListKey++;
        }

    else

        /*
         *  This symbol already has a LineList structure.  Malloc a new
         *  LineList and insert it at the head of the list.
         */

        {
        listPtr = (LineList *) malloc( sizeof(LineList) );
        listPtr->line = LDlinePtr;
        listPtr->next = (struct LineList *) LDendPtr->from;
        listPtr->prev = NULL;
        listPtr->key  = LineListKey++;
        LDendPtr->from->prev = (struct LineList *) listPtr;
        LDendPtr->from = listPtr;
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cancel_line()
*
*
* MODULE FUNCTION: This module erases the line drawn so far, frees the line structure,
*                  and clears the line field in the start symbol for the line.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

void  cancel_line()
{
    LineSeg *savPtr,
            *segPtr;


    /*
     *  NULL out symbol's Line pointer.
     */

    if ( LDside == NEXT_PTR )
        LDstartPtr->next = NULL;
    else if ( LDside == TRUE_PTR )
        LDstartPtr->Sym.IfSym.true_line = NULL;
    else
        LDstartPtr->Sym.IfSym.false_line = NULL;

    /*
     *  Clear and free the LineSegment structures.  Don't need to clear the cell
     *  map because it is filled in at the completion of the line.
     *
     *  Start clearing segments at the first segment.  The LDlinePtr points to
     *  the first segment in the list.
     */

    segPtr = LDlinePtr->line;
    while ( segPtr != NULL )
        {
        zap_line( segPtr );

        /*
         *  Save a pointer to the current line segment structure so we
         *  can free it.
         */

        savPtr = segPtr;
        segPtr = (LineSeg *) segPtr->next;
        free( savPtr );
        }

    /*
     *  Free the Line structure, all LineSegments should be gone.  The pointer
     *  in the symbol structure to this Line structure has already been cleared.
     */

    free( LDlinePtr );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  clear_cell_map_line()
*
*
* MODULE FUNCTION:
*
*   This routine clears the cells occupied by the parameter line segment.
*   Beware: the global var LDlinePtr must be set prior to using this function.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

void  clear_cell_map_line( segPtr )

    LineSeg *segPtr;

{
    LineList    *listPtr,
                *nextPtr,
                *prevPtr;
    int         cont     = TRUE,
                endPoint = FALSE,
                x,y;


    x = segPtr->cell_start_x;
    y = segPtr->cell_start_y;

    /*
     *  Loop through all the cells of the current line.
     */

    while ( cont )
        {

        /*
         *  Make sure the current cell is a line cell.  Lines run under
         *  symbols and symbols have priority, so there will be some
         *  cells of the line which are really symbol cells.
         */

        if ( Cell_Map[y][x].cell_type == LINE_CELL )
            {

            /*
             *  Set a pointer to this cell's LineList.
             */

            listPtr = Cell_Map[y][x].cell_entry.lines;

            /*
             *  Loop through the LineList looking for the Line entry which points to
             *  the current segment.
             */

            while ( listPtr != NULL )
                {

                /*
                 *  If the segment pointer in the current LineList structure points
                 *  to the current segment, remove this LineList pointer.
                 */

                if ( listPtr->line == LDlinePtr )
                    {

                    /*
                     *  If there is only one LineList structure, NULL the cell map.
                     */

                    if ((listPtr->next == NULL) && (listPtr->prev == NULL))
                        {
                        Cell_Map[y][x].cell_entry.lines = NULL;
                        Cell_Map[y][x].cell_type        = NONE;
                        free( listPtr );
                        break;
                        }

                    /*
                     *  If this is the LineList pointed to by the cell, (the head of the
                     *  linked list), then set a pointer to the second struct, update
                     *  the cell map and the prev pointer in the second LineList
                     *  strucutre.
                     */

                    if ( listPtr->prev == NULL )
                        {
                        nextPtr = (LineList *) listPtr->next;
                        Cell_Map[y][x].cell_entry.lines = nextPtr;
                        nextPtr->prev = NULL;
                        free( listPtr );
                        break;
                        }

                    /*
                     *  Otherwise, this is a LineList struct which is not the head of
                     *  the queue, so we don't need to handle the cell map directly.
                     */

                    prevPtr = (LineList *) listPtr->prev;
                    nextPtr = (LineList *) listPtr->next;
                    prevPtr->next = listPtr->next;
                    if ( nextPtr != NULL )
                        nextPtr->prev = (struct LineList *) prevPtr;
                    free( listPtr );
                    break;
                    }

                /*
                 *  This LineList doesn't point to the correct line, go
                 *  to the next LineList struct.
                 */

                else
                    {
                    if ( listPtr->next == NULL )
                        {
                        elog(1,"clear_cell_map - error - didn't find line pointer");
                        elog(1,"x %d  y %d - start x %d  start y %d", x, y,
                                                segPtr->cell_start_x,
```

```
                                        segPtr->cell_start_y );
                    elog(1,"end x %d  end y %d",segPtr->cell_end_x,
                                        segPtr->cell_end_y );
                    elog(1,"cell type %d", Cell_Map[y][x].cell_type );
                    exit( ERR );
                    }
                else
                    listPtr = (LineList *) listPtr->next;
                }
            }
        }

    /*
     *  Move to the next cell of the current line.
     */

    if ( segPtr->cell_end_x == segPtr->cell_start_x )
        y = (y < segPtr->cell_end_y) ? y+1 : y-1;
    else
        x = (x < segPtr->cell_end_x) ? x+1 : x-1;

    if ((x==segPtr->cell_end_x) && (y==segPtr->cell_end_y))
        endPoint = TRUE;
    else
        if ( endPoint )
            cont = FALSE;
    }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  clear_line()
*               draw_line()
*                set_line()
*                zap_line()
*
*
* MODULE FUNCTION:
*
*   clear_line : toggles the pixels in a line and so erases the line.
*   draw_line  : sets the pixels in a line and so draws the line.
*   set_line   : sets the pixels in a line segment using COPY instead of XOR
*   zap_line   : clears the pixels in a line and so erases the line.
*
*   NOTE:   the clear_line() and draw_line() functions ASSUME the Line Draw Graphics
*           Context (LDgc) GX function will be set to XOR.  All other functions which
*           modify the GX function must restore the GX function to XOR before exiting.
*           The zap_line() routine is a good example of a routine which modifies the
*           LDgc GX function and then restores it.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                                  Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

void  clear_line()
{
    XDrawLine( display, XtWindow(draw_area), LDgc, LDstartX, LDstartY, LDendX, LDendY );
}

void  draw_line()
{
    XDrawLine( display, XtWindow(draw_area), LDgc, LDstartX, LDstartY, LDendX, LDendY );
}

void  set_line()
{
    /*
     *  Set function to COPY and then set the foreground to black.  Then draw the
     *  line.  Then reset function back to XOR for clear_line() and draw_line()
     *  which ASSUME the function is XOR.  Also set the foreground to Black XOR with
     *  the background so when draw_line() XOR's with the foreground black lines
     *  will appear.
     */

    XSetFunction( display, LDgc, GXcopy );
    XSetForeground( display, LDgc, BlackPixel(display,DefaultScreen(display)) );

    XDrawLine( display, XtWindow(draw_area), LDgc, LDstartX, LDstartY, LDendX, LDendY );

    XSetFunction( display, LDgc, GXxor );
    XSetForeground( display, LDgc, BlackPixel(display,DefaultScreen(display)) ^
                    LDbackground );
}

void  zap_line( segPtr )

    LineSeg *segPtr;
```

```
{
    XSetFunction( display, LDgc, GXcopy );
    XSetForeground( display, LDgc, LDbackground );
    XDrawLine( display, XtWindow(draw_area), LDgc, segPtr->start_x, segPtr->start_y,
                                segPtr->end_x,   segPtr->end_y );
    XSetForeground( display, LDgc, BlackPixel(display, DefaultScreen(display)) ^
        LDbackground );
    XSetFunction( display, LDgc, GXxor );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  delete_line()
*
*
* MODULE FUNCTION:
*
*   This routine erases a line and clears the line fields in the start and destination
*   symbols.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                                  Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

void  delete_line()
{
    LineSeg     *savPtr,
                *segPtr;
    LineList    *listPtr,
                *nextPtr,
                *prevPtr;

    /*
     *  Set global line draw variables for the two symbols involved.
     */

    LDlistPtr   = Cell_Map[LDlineY][LDlineX].cell_entry.lines;

    if ((LDlistPtr == NULL) ||
        (LDlistPtr->line == NULL) ||
        (LDlistPtr->line->line == NULL) ||
        (LDlistPtr->line->to == NULL) ||
        (LDlistPtr->line->from == NULL) )
        {
        elog(1,"delete_line() - Internal error, no pointer");
        return;
        }

    LDlinePtr   = LDlistPtr->line;
    LDsegPtr    = LDlinePtr->line;
    LDendPtr    = (Symbol *) LDlinePtr->to;
    LDstartPtr  = (Symbol *) LDlinePtr->from;

    /*DEBUG*/
    elog(3,"delete line: deleting line connecting syms %d and %d\n",
        compute_label_index(LDstartPtr), compute_label_index(LDendPtr) );

    /*
     *  Clear Line pointer in start symbol.  If the start symbol was an
     *  IF, clear the line label also.
     */

    if ( LDstartPtr->next == LDlinePtr )
        {
        LDstartPtr->next = NULL;
        }
    else if ( LDstartPtr->Sym.IfSym.true_line == LDlinePtr )
        {
```

```
    LDstartPtr->Sym.IfSym.true_line = NULL;
    clear_text( draw_area, LDstartPtr->Sym.IfSym.true_x,
                LDstartPtr->Sym.IfSym.true_y, "TRUE" );
    }
else if ( LDstartPtr->Sym.IfSym.false_line == LDlinePtr )
    {
    LDstartPtr->Sym.IfSym.false_line = NULL;
    clear_text( draw_area, LDstartPtr->Sym.IfSym.false_x,
                LDstartPtr->Sym.IfSym.false_y, "FALSE" );
    }
else
    {
    user_ack("delete_line() - couldn't match line pointer");
    elog(1,"delete_line() - couldn't match line pointer");
    exit( ERR );
    }

/*
 *  Clear the "from" LineList structure in the end symbol.
 */

listPtr = LDendPtr->from;
while ( listPtr != NULL )
    {

    /*
     *  Check if the current LineList line pointer points to the
     *  line we want to delete.
     */

    if ( listPtr->line == LDlinePtr )
        {

        /*
         *  There is a single line entering this symbol and consequently there
         *  is only one LineList structure, so blow away the LineList structure
         *  and free its memory.
         */

        if ((listPtr->next == NULL) && (listPtr->prev == NULL))
            {
            LDendPtr->from = NULL;
            free( listPtr );
            break;
            }

        /*
         *  The LineList structure at the head of the list is the
         *  one that needs to be removed.
         */

        if ( listPtr->prev == NULL )
            {
            nextPtr = (LineList *) listPtr->next;
            LDendPtr->from = nextPtr;
            nextPtr->prev  = NULL;
            free( listPtr );
            break;
            }

        /*
         *  Otherwise, the LineList structure that we need to remove is
         *  past the head of the list.
         */

        prevPtr = (LineList *) listPtr->prev;
        nextPtr = (LineList *) listPtr->next;
        prevPtr->next = listPtr->next;
        if ( nextPtr != NULL )
            nextPtr->prev = (struct LineList *) prevPtr;
        free( listPtr );
        break;
        }

    /*
     *  The current LineList doesn't point to the correct line, go
     *  to the next LineList struct.
     */

    else
        {
        if ( listPtr->next == NULL )
            {
            elog(1,"delete_line() - internal error - didn't find line pointer");
            exit( ERR );
            }
        else
            listPtr = (LineList *) listPtr->next;
        }
    }

/*
 *  Clear and free the LineSegment structures.
 *
 *  Start clearing segments at the first segment.  The LDlinePtr points to
 *  the first segment in the list.
 */


/*
 *  get to last segment in line; erase arrows from that segment
 */

segPtr = LDsegPtr;

while ( segPtr != NULL )
    {
    if ( (segPtr->arrow_x) && (segPtr->arrow_y) )
        draw_arrows(segPtr, TRUE);
    segPtr = (LineSeg *)segPtr->next;
    }


segPtr = LDsegPtr;

while ( segPtr != NULL )
    {

    zap_line( segPtr );
    clear_cell_map_line( segPtr );

    /*
     *  Save a pointer to the current line segment structure so we
     *  can free it.
     */

    savPtr = segPtr;
    segPtr = (LineSeg *) segPtr->next;
```

```
        free( savPtr );
        }

    /*
     *  Free the Line structure, all LineSegments should be gone.  The pointer
     *  in the symbol structure to this Line structure has already been cleared.
     */
    free( LDlinePtr );

    /*
     *  status is automatically incomplete
     */

    upd_status(0);

    /*
     *  save is needed
     */

    SaveNeeded = TRUE;
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  delete_lines()
*
*
* MODULE FUNCTION: deletes all lines entering or leaving a symbol -
*                  used in remove_symbol and move_symbol
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

void  delete_lines( window )

    Widget window;

{
    Line *lineptr, *trueptr, *falseptr;
    int sym;
    void find_line_cell();

    for ( sym=0; sym<MAX_SYMBOLS; sym++ )
        {
        if ( Symbol_Map[sym].mycanvas == window )
            {

            /*
             *  delete "next" line
             */

            if ( Symbol_Map[sym].symbol_type == IF )
                {
                if ( trueptr = (Line *)Symbol_Map[sym].Sym.IfSym.true_line )
                    {
                    find_line_cell( trueptr->line );
                    delete_line();
                    }
                if ( falseptr = (Line *)Symbol_Map[sym].Sym.IfSym.false_line )
                    {
                    find_line_cell( falseptr->line );
                    delete_line();
                    }
                }
            else
                if ( (lineptr = (Line *)Symbol_Map[sym].next) != NULL )
                    {
                    find_line_cell( lineptr->line );
                    delete_line();
                    }

            /*
             *  delete lines entering symbol
             */

            while ( Symbol_Map[sym].from )
                {
                    if ( (lineptr = Symbol_Map[sym].from->line) != NULL )
                        {
                        /*
```

```
                    *  delete next "from" line
                    */

                   find_line_cell( lineptr->line );
                   delete_line();
                   }

                   /*
                    *  delete_line will advance the Symbol_Map.from
                    *  pointer.
                    */

               }
           return;
           }
       }

   elog(1,"delete_lines() - can't find symbol to delete lines\n");
   exit( ERR );
}
```

```
/*************************************<----->*************************************
*
* MODULE NAME:  draw_arrows()
*
*
* MODULE FUNCTION:  This routine draws the arrow at the end of a line with the
*                   correct orientation.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*************************************<----->*************************************/

int  draw_arrows(segptr, clear)

    LineSeg *segptr;
    int clear;

{
    if ( clear )
        {

        /*
         *  draw the arrows in the color of the background
         */

        XSetFunction( display, LDgc, GXcopy );
        XSetForeground( display, LDgc, LDbackground );
        }

    switch( segptr->orientation )
        {
        case RIGHT:
                XDrawLine( display, XtWindow(draw_area), LDgc,
                    segptr->arrow_x, segptr->arrow_y,
                    segptr->arrow_x - 8, segptr->arrow_y - 8 );
                XDrawLine( display, XtWindow(draw_area), LDgc,
                     segptr->arrow_x, segptr->arrow_y,
                    segptr->arrow_x - 8, segptr->arrow_y + 8 );
            break;

        case LEFT:
                XDrawLine( display, XtWindow(draw_area), LDgc,
                    segptr->arrow_x, segptr->arrow_y,
                    segptr->arrow_x + 8, segptr->arrow_y - 8 );
                XDrawLine( display, XtWindow(draw_area), LDgc,
                    segptr->arrow_x, segptr->arrow_y,
                    segptr->arrow_x + 8, segptr->arrow_y + 8 );
            break;
        case UP:
                XDrawLine( display, XtWindow(draw_area), LDgc,
                    segptr->arrow_x, segptr->arrow_y,
                    segptr->arrow_x - 8, segptr->arrow_y + 8 );
                XDrawLine( display, XtWindow(draw_area), LDgc,
                    segptr->arrow_x, segptr->arrow_y,
                    segptr->arrow_x + 8, segptr->arrow_y + 8 );
            break;
        case DOWN:
                XDrawLine( display, XtWindow(draw_area), LDgc,
```

```
                segptr->arrow_x, segptr->arrow_y,
                segptr->arrow_x - 8, segptr->arrow_y - 8 );
            XDrawLine( display, XtWindow(draw_area), LDgc,
                segptr->arrow_x, segptr->arrow_y,
                segptr->arrow_x + 8, segptr->arrow_y - 8 );
        break;
        }
    if ( clear )
        {
        XSetForeground( display, LDgc, BlackPixel(display,
            DefaultScreen(display)) ^ LDbackground );
        XSetFunction( display, LDgc, GXxor );
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  draw_whole_line()
*
* MODULE FUNCTION:
*
*   This routine draws all the segments of a line, as well as possible labels,
*   in either red or black.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  draw_whole_line(line, black)

    Line    *line;
    int     black;

{
    LineSeg     *segPtr;
    Symbol      *tempsym;

    erase_line( line );

    XSetFunction( display, LDgc, GXxor );
    if ( black )

        /*
         *  draw each line seg using black foreground.
         */

        XSetForeground( display, LDgc, BlackPixel(display, DefaultScreen(display)) ^
            LDbackground );

    else

        /*
         *  draw each line seg using Red foreground.
         */

        XSetForeground( display, LDgc, colors[Red] ^ LDbackground );

    tempsym = (Symbol *)line->from;
    if ( tempsym->symbol_type == IF )
        {
        if ( line == tempsym->Sym.IfSym.true_line )
            {
            XDrawString( display, XtWindow(draw_area), LDgc,
                tempsym->Sym.IfSym.true_x,  tempsym->Sym.IfSym.true_y,
                "TRUE", strlen("TRUE") );
            }
        else if ( line == tempsym->Sym.IfSym.false_line )
            {
            XDrawString( display, XtWindow(draw_area), LDgc,
                tempsym->Sym.IfSym.false_x,  tempsym->Sym.IfSym.false_y,
                "FALSE", strlen("FALSE") );
            }
        else
            {
```

```
            user_ack("if sym, but parameter line isnt true or false");
            exit(0);
            }
        }

    segPtr = line->line;
    while ( segPtr != NULL )
        {
        LDstartX = segPtr->start_x;
        LDstartY = segPtr->start_y;
        LDendX = segPtr->end_x;
        LDendY = segPtr->end_y;
        draw_line();
        if ( !segPtr->next )
            {

            /*
             *  last segment in line; draw arrows
             */

            draw_arrows( segPtr, FALSE );
            }
        segPtr = (LineSeg *) segPtr->next;
        }

    /*
     *  reset gc foreground to black.
     */

    if ( !black )
        XSetForeground( display, LDgc, BlackPixel(display, DefaultScreen(display)) ^
            LDbackground );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  end_Line()
*
*
* MODULE FUNCTION:  This routine completes the line structures in the start and
*                   destination symbols, sets the cells in the line, and draws
*                   the arrows and labels.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

void  end_Line()
{
    LineSeg *segPtr, *arrowPtr;
    char    string[6];
    int     x,y;
    LineSeg *get_good_cell_pt();


    /*
     *  Complete the Line and LineSegment structures.
     */

    LDlinePtr->to   = (struct Symbol *) LDendPtr;

    LDsegPtr->cell_end_x    = LDendX / CELL_SIZE;
    LDsegPtr->cell_end_y    = LDendY / CELL_SIZE;
    LDsegPtr->end_x         = LDendX;
    LDsegPtr->end_y         = LDendY;
    if (LDendX > LDstartX)
        LDsegPtr->orientation = RIGHT;
    else if (LDendX < LDstartX)
        LDsegPtr->orientation = LEFT;
    else if (LDendY < LDstartY)
        LDsegPtr->orientation = UP;
    else
        LDsegPtr->orientation = DOWN;


    /*
     *  Add "from" line to end symbol.
     */

    add_from_line();

    /*
     *  Make sure that last pt in line is of LINE type
     */

    arrowPtr = get_good_cell_pt( LDsegPtr );

    /*
     *  Update the cell map for each of the line segments which make
     *  up this line.
     */
```

```
segPtr = LDsegPtr;

while ( segPtr != NULL )
    {
    set_cell_map_line( segPtr, LDlinePtr );
    segPtr = (LineSeg *) segPtr->prev;
    }

/*
 *  Determine where to put the arrows, 2nd parameter would be true to
 *  delete arrows
 */

draw_arrows(arrowPtr, FALSE);

/*
 *  if status is complete, update status message
 */

if ( complete(0, LINES_AND_EXPR) == -1 )
    upd_status( 1 );

/*
 *  Element needs to be saved.
 */

SaveNeeded = TRUE;

/*
 *  If the current line is part of an IF symbol, label the current
 *  path.
 */

if ( LDside == NEXT_PTR )
    return;

/*
 *  Determine which side of the IF the user is connecting.
 */

if ( LDside == TRUE_PTR )
    {
    strcpy( string, "TRUE" );
    segPtr = (LineSeg *) LDstartPtr->Sym.IfSym.true_line->line;
    }
else
    {
    strcpy( string, "FALSE" );
    segPtr = (LineSeg *) LDstartPtr->Sym.IfSym.false_line->line;
    }

/*
 *  Determine where to put the label.
 */

if ( segPtr->end_x > segPtr->start_x )
    {
    x = LDstartPtr->ulcx + LDstartPtr->width  + 2;
    y = segPtr->start_y - 10;
    }
else if ( segPtr->end_x < segPtr->start_x )
    {
    x = LDstartPtr->ulcx - 40;
    y = segPtr->start_y - 10;
    }
else if ( segPtr->end_y > segPtr->start_y )
    {
    x = segPtr->start_x + 10;
    y = LDstartPtr->ulcy + LDstartPtr->height + 10;
    }
else if ( segPtr->end_y < segPtr->start_y )
    {
    x = segPtr->start_x + 10;
    y = LDstartPtr->ulcy - 5;
    }

/*
 *  Draw label.
 */

XDrawString( display, XtWindow(draw_area), LDgc, x, y, string, strlen(string) );

/*
 *  Record label in symbol structure.
 */

if ( LDside == TRUE_PTR )
    {
    LDstartPtr->Sym.IfSym.true_x = x;
    LDstartPtr->Sym.IfSym.true_y = y;
    }
else
    {
    LDstartPtr->Sym.IfSym.false_x = x;
    LDstartPtr->Sym.IfSym.false_y = y;
    }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  erase_line()
*
* MODULE FUNCTION:
*
*   This routine clears the pixels in the specified line without changing the
*   fields in the start or end symbols.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  erase_line(line)

    Line    *line;

{
    LineSeg *segPtr;
    Symbol  *tempsym;

    /*
     *  get to last segment in line; erase arrows from that segment
     */

    segPtr = line->line;

    while ( segPtr != NULL )
        {
        if ( !segPtr->next )
            draw_arrows( segPtr, TRUE );
        segPtr = (LineSeg *)segPtr->next;
        }

    /*
     *  if this is an if line, clear label
     */

    tempsym = (Symbol *)line->from;
    if (tempsym->symbol_type == IF)
        {
        if ( line == tempsym->Sym.IfSym.true_line )
            {
            clear_text( draw_area, tempsym->Sym.IfSym.true_x,
                tempsym->Sym.IfSym.true_y, "TRUE" );
            }
        else if ( line == tempsym->Sym.IfSym.false_line )
            {
            clear_text( draw_area, tempsym->Sym.IfSym.false_x,
                tempsym->Sym.IfSym.false_y, "FALSE" );
            }
        else
            {
            user_ack("if sym, but parameter line isnt true or false");
            exit(0);
            }
        }

    /*
     *  clear each line seg.
     */

    segPtr = line->line;
    while ( segPtr != NULL )
        {
        LDstartX = segPtr->start_x;
        LDstartY = segPtr->start_y;
        LDendX = segPtr->end_x;
        LDendY = segPtr->end_y;
        zap_line( segPtr );
        segPtr = (LineSeg *) segPtr->next;
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  find_line_cell()
*
*
* MODULE FUNCTION:  Given a line segment, this routine finds the first cell in it
*   that is of type LINE_CELL.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  find_line_cell( lineseg )

    LineSeg *lineseg;

{

    int cellx, celly;

    while ( lineseg )
        {
        cellx = lineseg->cell_start_x;
        celly = lineseg->cell_start_y;
        while ( still_on_line(cellx, celly, lineseg) )
            {
            if ( Cell_Map[celly][cellx].cell_type == LINE_CELL )
                {
                LDlineX = cellx;
                LDlineY = celly;
                return;
                }
            else
                {
                if ( lineseg->orientation == UP )
                        celly--;
                else if ( lineseg->orientation == DOWN )
                        celly++;
                else if ( lineseg->orientation == RIGHT )
                        cellx++;
                else if ( lineseg->orientation == LEFT )
                        cellx--;
                }
            }
        lineseg = (LineSeg *)lineseg->next;
        }
    user_ack("find line cell: couldnt find line cell");
    exit(0);
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  free_cellmap_linelists()
*
*
* MODULE FUNCTION:  This routine frees all the linelists in all of the cells.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  free_cellmap_linelists()
{
    LineList    *currPtr,
                *nextPtr;
    register int i,j;


    for ( i=0; i<CELL_ROWS; i++ )
        for (j=0; j<CELL_COLS; j++ )
            if ( Cell_Map[i][j].cell_type == LINE_CELL )
                {
                currPtr = Cell_Map[i][j].cell_entry.lines;
                while ( currPtr != NULL )
                    {
                    nextPtr = (LineList *) currPtr->next;
                    free( currPtr );
                    currPtr = nextPtr;
                    }
                Cell_Map[i][j].cell_entry.lines = NULL;
                }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  free_lines()
*
*
* MODULE FUNCTION:  This routine deallocates all the space for all the lines
*                   in the Symbol_Map array.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  free_lines()
{
    int sym;


    for ( sym=0; sym<MAX_SYMBOLS; sym++ )
        {
        if ( Symbol_Map[sym].symbol_type != NONE )
            {
            if ( Symbol_Map[sym].next != NULL )
                {
                free_segment( Symbol_Map[sym].next->line );
                free( Symbol_Map[sym].next );
                }
            if ( Symbol_Map[sym].from != NULL )
                free_list( Symbol_Map[sym].from );

            if ( Symbol_Map[sym].symbol_type == IF)
                {
                if ( Symbol_Map[sym].Sym.IfSym.false_line != NULL )
                    {
                    free_segment( Symbol_Map[sym].Sym.IfSym.false_line->line );
                    free( Symbol_Map[sym].Sym.IfSym.false_line );
                    }
                if ( Symbol_Map[sym].Sym.IfSym.true_line != NULL )
                    {
                    free_segment( Symbol_Map[sym].Sym.IfSym.true_line->line );
                    free( Symbol_Map[sym].Sym.IfSym.true_line );
                    }
                }
            }
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  free_list()
*
*
* MODULE FUNCTION:  This routine frees all the lines in the parameter line list.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

void  free_list( listPtr )

    LineList *listPtr;

{
    LineList    *currList,
                *nextList;

    currList = listPtr;
    while ( currList != NULL )
        {
        nextList = (LineList *) currList->next;
        free( currList );
        currList = nextList;
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  free_segment()
*
*
* MODULE FUNCTION:  This routine frees the space in all the segments of a line.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

void  free_segment( segPtr )

    LineSeg *segPtr;

{
    LineSeg *currSeg,
            *nextSeg;

    currSeg = segPtr;
    while ( currSeg != NULL )
        {
        nextSeg = (LineSeg *) currSeg->next;
        free( currSeg );
        currSeg = nextSeg;
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  get_good_cell_pt()
*
*
* MODULE FUNCTION:  This routine sets the endpoint of the parameter segment to be
*                   the last cell outside the destination symbol.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

LineSeg  *get_good_cell_pt(ldlineseg)

    LineSeg *ldlineseg;

{
    LineSeg *lineseg = (LineSeg *)ldlineseg;
    int done = 0;


    while ( (lineseg) && (!(done)) )
        {

        /*
         *  find last segment in this line whose startpt is outside LDendPtr
         */

        if ( lineseg->cell_end_x == lineseg->cell_start_x )

            /*
             *  vertical line
             */

            if ( lineseg->cell_start_y > lineseg->cell_end_y )

                {

                /*
                 *  endpt is higher than startpt
                 */

                if ( lineseg->start_y > LDendPtr->ulcy + LDendPtr->height )
                    {
                    lineseg->cell_end_y = LDendPtr->cell_y + LDendPtr->cell_height;
                    lineseg->orientation = UP;
                    done++;
                    }
                }

            else
                {

                /*
                 *
                 *  endpt is lower than startpt
                 */

                if ( lineseg->start_y < LDendPtr->ulcy )
```

```
                {
                lineseg->cell_end_y = LDendPtr->cell_y ;
                lineseg->orientation = DOWN;
                done++;
                }
            }
        else
            /*
             *  horizontal line
             */

            if ( lineseg->cell_start_x > lineseg->cell_end_x )

                {

                /*
                 *  line is moving to the left
                 */

                if ( lineseg->start_x > LDendPtr->ulcx + LDendPtr->width )
                    {
                    lineseg->cell_end_x = LDendPtr->cell_x + LDendPtr->cell_width ;
                    lineseg->orientation = LEFT;
                    done++;
                    }
                }
            else
                {

                /*
                 *  line is moving to the right
                 */

                if ( lineseg->start_x < LDendPtr->ulcx )
                    {
                    lineseg->cell_end_x = LDendPtr->cell_x;
                    lineseg->orientation = RIGHT;
                    done++;
                    }
                }

        if ( !(done) )
            lineseg = (LineSeg *)lineseg->prev;
        }

    switch ( lineseg->orientation )
        {
        case UP:
            lineseg->arrow_x = lineseg->end_x;
            lineseg->arrow_y = LDendPtr->ulcy + LDendPtr->height - 1;
            break;
        case DOWN:
            lineseg->arrow_x = lineseg->end_x;
            lineseg->arrow_y = LDendPtr->ulcy - 1;
            break;
        case RIGHT:
            lineseg->arrow_y = lineseg->end_y;
            lineseg->arrow_x = LDendPtr->ulcx - 1;
            break;
        case LEFT:
            lineseg->arrow_y = lineseg->end_y;
            lineseg->arrow_x = LDendPtr->ulcx + LDendPtr->width - 1;
            break;
        }

    return lineseg;
}
```

```
/*************************************<----->*************************************
*
* MODULE NAME:  mid_Line()
*
*
* MODULE FUNCTION:  This routine allocates a new segment pointer when the user
*                   sets an 'elbow' in a line.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*************************************<----->*************************************/

void  mid_Line()
{
    LineSeg *new;

    /*
     *  Now that we know the end of the current line segment,
     *  complete the structure.
     */

    LDsegPtr->cell_end_x    = LDendX / CELL_SIZE;
    LDsegPtr->cell_end_y    = LDendY / CELL_SIZE;
    LDsegPtr->end_x         = LDendX;
    LDsegPtr->end_y         = LDendY;

    if ( LDendX > LDstartX )
        LDsegPtr->orientation = RIGHT;
    else if ( LDendX < LDstartX )
        LDsegPtr->orientation = LEFT;
    else if ( LDendY < LDstartY )
        LDsegPtr->orientation = UP;
    else
        LDsegPtr->orientation = DOWN;


    /*
     *  The two pointers below are initialized differently to make
     *  the compiler happy and to avoid warning messages.
     *
     *  Allocate a new line segment structure and set the pointer in
     *  the previous line segment to point to the new segment.
     */

    new             = (LineSeg *) malloc( sizeof(LineSeg) );
    LDsegPtr->next  = (struct LineSeg *) new;

    /*
     *  Start a new line segment structure.
     */

    new->cell_start_x       = LDsegPtr->cell_end_x;
    new->cell_start_y       = LDsegPtr->cell_end_y;
    new->start_x            = LDsegPtr->end_x;
    new->start_y            = LDsegPtr->end_y;
    new->arrow_x            = 0;
    new->arrow_y            = 0;
    new->prev               = (struct LineSeg *) LDsegPtr;
    new->next               = NULL;
    new->key                = LineSegKey++;

    LDsegPtr                = new;
}
```

```
/*************************************<----->*************************************
*
* MODULE NAME:  prompt_true_false()
*
*
* MODULE FUNCTION:  This module determines which of the true or false lines the
*                   user wants to draw.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*************************************<----->*************************************/

int  prompt_true_false( parentcanvas )

    Widget parentcanvas;

{
    XmString    templ_tcs,
                temp2_tcs;
    Arg         args[1];
    int         i;


    /*
     *  Create TRUE/FALSE strings for Ask popup.
     */

    XmString t_tcs = XmStringCreate( "TRUE",  XmSTRING_DEFAULT_CHARSET );
    XmString f_tcs = XmStringCreate( "FALSE", XmSTRING_DEFAULT_CHARSET );

    /*
     *  store current Ask strings
     */

    XtSetArg( args[0], XmNlabelString, &templ_tcs );
    XtGetValues( btn_ask_y, args, 1 );

    XtSetArg( args[0], XmNlabelString, &temp2_tcs );
    XtGetValues( btn_ask_n, args, 1 );

    /*
     *  Set new strings.
     */

    XtSetArg( args[0], XmNlabelString, t_tcs );
    XtSetValues( btn_ask_y, args, 1 );

    XtSetArg( args[0], XmNlabelString, f_tcs );
    XtSetValues( btn_ask_n, args, 1 );

    /*
     *  Set new size - minimum button size.
     */

    XtSetArg( args[0], XmNwidth, MIN_BTN_WIDTH );
    XtSetValues( btn_ask_y, args, 1 );
    XtSetValues( btn_ask_n, args, 1 );

    /*
     *  Display the popup and ask the user if they want to connect the TRUE or
     *  FALSE side.
     */

    if ( ask(tf_string) )
        i =  TRUE_PTR ;
    else
        i = FALSE_PTR;

    XmStringFree( t_tcs );
    XmStringFree( f_tcs );

    /*
     *  restore previous ask strings
     */

    XtSetArg( args[0], XmNlabelString, templ_tcs );
    XtSetValues( btn_ask_y, args, 1 );

    XtSetArg( args[0], XmNlabelString, temp2_tcs );
    XtSetValues( btn_ask_n, args, 1 );

    XtSetArg( args[0], XmNwidth, MIN_BTN_WIDTH );
    XtSetValues( btn_ask_y, args, 1 );
    XtSetValues( btn_ask_n, args, 1 );

    return( i );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  set_cell_map_line()
*
*
* MODULE FUNCTION:
*
*   This routine creates the LineList structures necessary to record the cells
*   utilized by a line.  This routine only creates LineList structures, it does
*   not create Line structures or LineSeg structures.
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

void  set_cell_map_line( segPtr, linePtr )

    LineSeg *segPtr;
    Line    *linePtr;

{
    LineList    *listPtr;
    Cell        *cell;
    int         cont     = TRUE,
                endPoint = FALSE,
                x,y;

    x = segPtr->cell_start_x;
    y = segPtr->cell_start_y;

    while ( cont )
        {

        cell = &(Cell_Map[y][x]);

        /*
         *  Currently, the lines start and end in the middle of symbols,
         *  if this cell belongs to a symbol, forget it, let the symbol
         *  have it.
         */

        if ( cell->cell_type != SYMBOL_CELL )
            {
            cell->cell_type = LINE_CELL;

            /*
             *  If there is not an existing line list, then malloc a new one.  Set
             *  the next pointer in the new line list to NULL, set the line pointer,
             *  then set the cell to point to the line list strucutre.
             */

            if ( cell->cell_entry.lines == NULL )
                {
                cell->cell_entry.lines = (LineList *) malloc( sizeof(LineList) );
                cell->cell_entry.lines->next = NULL;
                cell->cell_entry.lines->prev = NULL;
                cell->cell_entry.lines->line = linePtr;
                cell->cell_entry.lines->key  = LineListKey++;
                }

            /*
             *  This cell has a line list structure already.  Malloc a new line
             *  list entry and set its next pointer equal to the pointer in the
             *  first list entry.  Add the new cell at the head of the list
             *  pushing everyone behind the new list.
             *
             *  Use the long form of pointer casting (struct LineList *) instead
             *  of the typedef (LineList *) to make the compiler happy.
             */

            else
                {
                listPtr = (LineList *) malloc( sizeof(LineList) );
                listPtr->next = (struct LineList *) cell->cell_entry.lines;
                listPtr->prev = NULL;
                listPtr->line = linePtr;
                listPtr->key  = LineListKey++;
                cell->cell_entry.lines->prev = (struct LineList *) listPtr;
                cell->cell_entry.lines = listPtr;
                }
            }

        /*
         *  Goto the next cell in this line segment.
         */

        if ( segPtr->cell_end_x == segPtr->cell_start_x )
            y = (y < segPtr->cell_end_y) ? y+1 : y-1;
        else
            x = (x < segPtr->cell_end_x) ? x+1 : x-1;

        if ( (x==segPtr->cell_end_x) && (y==segPtr->cell_end_y) )
            endPoint = TRUE;
        else
            if ( endPoint )
                cont = FALSE;
        }
}
```

```
/*************************************<----->*************************************
*
* MODULE NAME:  start_Line()
*
*
* MODULE FUNCTION:  This module creates the new structures needed to start a line
*                   and initializes the fields in the start symbol.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*************************************<----->*************************************/

void  start_Line( side )

    int side;

{
    /*
     *  Get a new Line struct.
     */

    LDlinePtr = (Line *) malloc( sizeof(Line) );

    /*
     *  Set "next" pointer in symbol.
     */

    switch ( side )
        {
        case FALSE_PTR : LDstartPtr->Sym.IfSym.false_line = LDlinePtr;
                         break;
        case NEXT_PTR :  LDstartPtr->next = LDlinePtr;
                         break;
        case TRUE_PTR :  LDstartPtr->Sym.IfSym.true_line = LDlinePtr;
                         break;
        }

    /*
     *  Load up structure.
     */

    LDlinePtr->line = LDsegPtr = (LineSeg *) malloc( sizeof(LineSeg) );
    LDlinePtr->from = (struct Symbol *) LDstartPtr;
    LDlinePtr->to   = NULL;
    LDlinePtr->key  = LineKey++;

    LDsegPtr->cell_start_x  = LDstartX / CELL_SIZE;
    LDsegPtr->cell_start_y  = LDstartY / CELL_SIZE;
    LDsegPtr->start_x       = LDstartX;
    LDsegPtr->start_y       = LDstartY;
    LDsegPtr->arrow_x       = 0;
    LDsegPtr->arrow_y       = 0;
    LDsegPtr->prev          = NULL;
    LDsegPtr->next          = NULL;
    LDsegPtr->key           = LineSegKey++;
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  still_on_line()
*
*
* MODULE FUNCTION:  This module determines if the given point is on the given segment.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

int  still_on_line(x, y, lineseg)

    int x, y;
    LineSeg *lineseg;

{

    switch ( lineseg->orientation )
        {
        case UP:
            if ( y >= lineseg->cell_end_y )
                return 1;
            else return 0;
        case DOWN:
            if ( y <= lineseg->cell_end_y )
                return 1;
            else return 0;
        case RIGHT:
            if ( x <= lineseg->cell_end_x )
                return 1;
            else return 0;
        case LEFT:
            if ( x >= lineseg->cell_end_x )
                return 1;
            else return 0;
        }

}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  valid_line()
*
*
* MODULE FUNCTION:  This module determines if the drawn line segment passes through
*                   a symbol.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  valid_line( type )

    int type;

{
    int cell_x, cell_y,
        cont     = TRUE,
        endPoint = FALSE,
        x,y;


    switch ( type )
        {
        case START_SEGMENT :
        case MID_SEGMENT :

            x = LDstartX;
            y = LDstartY;
            while ( cont )
                {
                cell_x = x / CELL_SIZE;
                cell_y = y / CELL_SIZE;

                if ( Cell_Map[cell_y][cell_x].cell_type == SYMBOL_CELL )
                    {
                    if ( Cell_Map[cell_y][cell_x].cell_entry.symbol != LDstartPtr )
                        return( ERR );
                    }

                if ( LDendX == LDstartX )
                    y = (y < LDendY) ? y+1 : y-1;
                else
                    x = (x < LDendX) ? x+1 : x-1;

                if ((x==LDendX) && (y==LDendY))
                    endPoint = TRUE;
                else
                    if ( endPoint )
                        cont = FALSE;
                }
            break;

        case END_SEGMENT :

            x = LDstartX;
            y = LDstartY;
            while ( cont )
```

```
            {
            cell_x = x / CELL_SIZE;
            cell_y = y / CELL_SIZE;

            if ( Cell_Map[cell_y][cell_x].cell_type == SYMBOL_CELL )
                {
                if ((Cell_Map[cell_y][cell_x].cell_entry.symbol != LDstartPtr) &&
                    (Cell_Map[cell_y][cell_x].cell_entry.symbol != LDendPtr  ))
                    return( ERR );
                }

            if ( LDendX == LDstartX )
                y = (y < LDendY) ? y+1 : y-1;
            else
                x = (x < LDendX) ? x+1 : x-1;

            if ((x==LDendX) && (y==LDendY))
                endPoint = TRUE;
            else
                if ( endPoint )
                    cont = FALSE;
            }

        break;
    }

return( OK );
}
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    lines.h
*
* FILE FUNCTION:
*
*   This file contains the constants and variables used to maintain the lines
*   between symbols.  These contants and variables are primarily maintained in
*   the file: lines.c
*
*
* SPECIFICATION DOCUMENTS:
*
*   /home/project/3531/Docu/GCB.spec.doc
*
*
* FILE MODULES:
*
*   N/A
*
*****************************************<----->*****************************************/

/*
 *  Constants used in lines.c
 */

#define START_SEGMENT   0
#define MID_SEGMENT     1
#define END_SEGMENT     2

#define FALSE_PTR       1
#define NEXT_PTR        2
#define TRUE_PTR        3

/*
 *  Globals.
 */

Drawable    LDdraw;
GC          LDgc;
Pixel       LDforeground, LDbackground;

Line        *LDlinePtr;
LineList    *LDlistPtr;
LineSeg     *LDsegPtr;

Symbol      *LDendPtr,          /* ptr to end   symbol of current line          */
            *LDstartPtr;        /* ptr to start symbol of current line          */

int         LDendX,
            LDendY,
            LDlineX,
            LDlineY,
            LDoldX,
            LDoldY,
            LDside,             /* indicates which part of an IF to connect     */
            LDstartX,
            LDstartY,
            LDtype;             /* start segment, mid segment, or end segment   */

/*
 *  Function prototypes.
 */

int     prompt_true_false   (),
        valid_line          ();

void    add_from_line       (),
        cancel_line         (),
        clear_cell_map_line (),
        clear_line          (),
        draw_line           (),
        end_line            (),
        find_line_cell      (),
        free_lines          (),
        free_list           (),
        free_segment        (),
        mid_line            (),
        set_cell_map_line   (),
        set_line            (),
        start_line          (),
        zap_line            ();
```

```
/*
 *  Include file used to manage matrices.
 */

#define ADD         0
#define SUBTRACT    1

#define INT         1
#define FLOAT       2
#define DOUBLE      3
#define SHORT       4
#define UNSIGN      5
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    menu.h
*
* FILE FUNCTION:
*
*   This file contains the program constants, variable declarations and function
*   prototypes for the routines in expr_menu.c and sub_menus.c and other various
*   source files which create and maintain menus and selection palettes.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /home/project/3531/Docu/GCB.spec.doc
*
*
* FILE MODULES:
*
*   N/A
*
*****************************************<----->*****************************************/


#define Banded                  1
#define Default                 2
#define Scrolled                3
#define Zooming                 4

#define NORMAL                  0
#define REDUCED                 1

#define CBR_COMP_TYPE           0
#define CBR_COMP_TYPE_DONE      1
#define CBR_COMP_TYPE_CANCEL    2

#define COMP_DELETE_APPLY       99

#define M_SHOW                  1       /* display the matrix menu          */
#define M_ADD                   2
#define M_SUB                   3
#define M_MULT                  4
#define M_IDENT                 5
#define M_INV                   6
#define M_TRAN                  7
#define M_CROSS                 8
#define M_DOT                   9

#define T_SHOW                  1       /* display the trig menu            */
#define T_COS                   2
#define T_ACOS                  3
#define T_TAN                   4
#define T_ATAN                  5
#define T_SIN                   6
#define T_ASIN                  7

/*
 *  Variables.
 */

int BStartx,
    BStarty,
    Copies,
    PrintScale,
    old_lrcx,
    old_lrcy,
    old_ulcx,
    old_ulcy;


/*
 *  Function prototypes.
 */

XtCallbackProc  cbr_if(),
                cbr_matrix(),
                cbr_palette(),
                cbr_print(),
                cbr_printset(),
                cbr_quaternion(),
                cbr_trig();

int             make_ps_file();

void            delete_box();
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    next_inputs.h
*
* FILE FUNCTION:
*
*   Header file for the DECISION and SET logical expression input buttons.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /home/project/3531/Docu/GCB.spec.doc
*
*
* FILE MODULES:
*
*   N/A
*
*****************************************<----->*****************************************/

#define NUM_BUTTONS 37

#define MSID        0
#define GLOBAL      1
#define LOCAL       2
#define NUMBER      3
#define STRING      4
#define AND         5
#define OR          6
#define NOT         7
#define BITAND      8
#define BITOR       9
#define BITXOR      10
#define EQ          11
#define NE          12
#define LE          13
#define GE          14
#define LT          15
#define GT          16
#define PLUS        17
#define MINUS       18
#define MUL         19
#define DIV         20
#define LPAR        21
#define RPAR        22
#define COMMA       23
#define SHIFTL      24
#define SHIFTR      25
#define PI          26
#define EXP         27
#define POWER       28
#define SQRT        29
#define LOG         30
#define NLOG        31
#define TRIG        32
#define QUAT        33
#define DEF_FN      34
#define MAT_FN      35
#define WSG         36

/*
 *  Global variables used in next_inputs.c
 */
int     valid_points[NUM_BUTTONS],
        ParenCount,
        WhereAmI;

/*
 *  Global variables used in next_inputs.c
 */

void    next_inputs(),
        invalidate_buttons();
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    pixmaps.h
*
* FILE FUNCTION:
*
*   This file contains the arrays which define the various pixmaps used within the
*   Graphical Comp Builder.  The pixmaps for the palette area (BEGIN, END, SET
*   and other symbols) and the pixmaps for the user_ack() and ask() popups are
*   defined in this file.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /home/project/3531/Docu/GCB.spec.doc
*
*
* FILE MODULES:
*
*   N/A
*
*****************************************<----->*****************************************/

#define activate_width 64
#define activate_height 64
static char activate_bits[] = {
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0xfe, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x3f,
   0x12, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x24, 0x12, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x24, 0x12, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x24,
   0x12, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x24, 0x12, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x24, 0x12, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x24,
   0x12, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x24, 0x12, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x24, 0x12, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x24,
   0x12, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x24, 0x12, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x24, 0x12, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x24,
   0x12, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x24, 0x12, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x24, 0x12, 0x08, 0xef, 0x15, 0x84, 0xf0, 0x3d, 0x24,
   0x12, 0x08, 0x41, 0x14, 0x84, 0x40, 0x04, 0x24, 0x12, 0x08, 0x41, 0x14,
   0x84, 0x40, 0x04, 0x24, 0x12, 0x14, 0x41, 0x24, 0x42, 0x41, 0x04, 0x24,
   0x12, 0x14, 0x41, 0x24, 0x42, 0x41, 0x04, 0x24, 0x12, 0x14, 0x41, 0x24,
   0x42, 0x41, 0x04, 0x24, 0x12, 0x14, 0x41, 0x24, 0x42, 0x41, 0x04, 0x24,
   0x12, 0x14, 0x41, 0x44, 0x42, 0x42, 0x04, 0x24, 0x12, 0x22, 0x41, 0x44,
   0x21, 0x42, 0x3c, 0x24, 0x12, 0x3e, 0x41, 0x44, 0xe1, 0x43, 0x04, 0x24,
   0x12, 0x22, 0x41, 0x44, 0x21, 0x42, 0x04, 0x24, 0x12, 0x22, 0x41, 0x44,
   0x21, 0x42, 0x04, 0x24, 0x12, 0x41, 0x41, 0x84, 0x10, 0x44, 0x04, 0x24,
   0x12, 0x41, 0x41, 0x84, 0x10, 0x44, 0x04, 0x24, 0x12, 0x41, 0x4f, 0x84,
   0x10, 0x44, 0x3c, 0x24, 0x12, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x24,
   0x12, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x24, 0x12, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x24, 0x12, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x24,
   0x12, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x24, 0x12, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x24, 0x12, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x24,
   0x12, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x24, 0x12, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x24, 0x12, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x24,
   0x12, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x24, 0xfe, 0xff, 0xff, 0xff,
   0xff, 0xff, 0xff, 0x3f, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};



#define begin_width 64
#define begin_height 64
static char begin_bits[] = {
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0xf8, 0xff, 0xff, 0xff, 0xff, 0x0f, 0x00, 0x00, 0x06, 0x00, 0x00,
   0x00, 0x00, 0x30, 0x00, 0x80, 0x01, 0x00, 0x00, 0x00, 0x00, 0xc0, 0x00,
   0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x40, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x01, 0x20, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x02,
   0x20, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x02, 0x10, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x04, 0x10, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x04,
   0x08, 0x00, 0x7c, 0xbe, 0x5f, 0x21, 0x00, 0x08, 0x08, 0x00, 0x84, 0x82,
   0x40, 0x21, 0x00, 0x08, 0x08, 0x00, 0x84, 0x82, 0x40, 0x23, 0x00, 0x08,
   0x08, 0x00, 0x84, 0x82, 0x40, 0x23, 0x00, 0x08, 0x04, 0x00, 0x84, 0x82,
   0x40, 0x23, 0x00, 0x10, 0x04, 0x00, 0x84, 0x82, 0x40, 0x23, 0x00, 0x10,
   0x04, 0x00, 0x84, 0x82, 0x40, 0x25, 0x00, 0x10, 0x04, 0x00, 0x84, 0x82,
   0x40, 0x25, 0x00, 0x10, 0x04, 0x00, 0x84, 0x82, 0x40, 0x25, 0x00, 0x10,
   0x04, 0x00, 0xfc, 0x8e, 0x5f, 0x29, 0x00, 0x10, 0x04, 0x00, 0x84, 0x82,
   0x50, 0x29, 0x00, 0x10, 0x04, 0x00, 0x84, 0x82, 0x50, 0x29, 0x00, 0x10,
   0x04, 0x00, 0x84, 0x82, 0x50, 0x29, 0x00, 0x10, 0x04, 0x00, 0x84, 0x82,
   0x50, 0x31, 0x00, 0x10, 0x04, 0x00, 0x84, 0x82, 0x50, 0x31, 0x00, 0x10,
   0x08, 0x00, 0x84, 0x82, 0x50, 0x31, 0x00, 0x08, 0x08, 0x00, 0x84, 0x82,
   0x50, 0x31, 0x00, 0x08, 0x08, 0x00, 0x84, 0x82, 0x50, 0x21, 0x00, 0x08,
   0x08, 0x00, 0x7c, 0xbe, 0x5f, 0x21, 0x00, 0x08, 0x10, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x04, 0x10, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x04,
   0x20, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x02, 0x20, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x02, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01,
   0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x80, 0x03, 0x00, 0x00,
   0x00, 0x00, 0xe0, 0x00, 0x00, 0x06, 0x00, 0x00, 0x00, 0x00, 0x30, 0x00,
   0x00, 0xfc, 0xff, 0xff, 0xff, 0xff, 0x1f, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};



#define box_width 16
#define box_height 16
static char box_bits[] = {
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
    0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff};
```

```
#define call_width 64
#define call_height 64
static char call_bits[] = {
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xc0, 0xff, 0xff,
   0xff, 0xff, 0x07, 0x00, 0x00, 0x30, 0x00, 0x00, 0x00, 0x00, 0x18, 0x00,
   0x00, 0x08, 0x00, 0x00, 0x00, 0x00, 0x20, 0x00, 0x00, 0x06, 0x00, 0x00,
   0x00, 0x00, 0xc0, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01,
   0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x02, 0x80, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x02, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x04,
   0x20, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x08, 0x20, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x08, 0x10, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x10,
   0x10, 0x00, 0x7f, 0x10, 0x04, 0x08, 0x00, 0x10, 0x10, 0x00, 0x01, 0x10,
   0x04, 0x08, 0x00, 0x10, 0x08, 0x00, 0x01, 0x10, 0x04, 0x08, 0x00, 0x20,
   0x08, 0x00, 0x01, 0x10, 0x04, 0x08, 0x00, 0x20, 0x08, 0x00, 0x01, 0x28,
   0x04, 0x08, 0x00, 0x20, 0x08, 0x00, 0x01, 0x28, 0x04, 0x08, 0x00, 0x20,
   0x04, 0x00, 0x01, 0x28, 0x04, 0x08, 0x00, 0x40, 0x04, 0x00, 0x01, 0x28,
   0x04, 0x08, 0x00, 0x40, 0x04, 0x00, 0x01, 0x28, 0x04, 0x08, 0x00, 0x40,
   0x04, 0x00, 0x01, 0x28, 0x04, 0x08, 0x00, 0x40, 0x04, 0x00, 0x01, 0x28,
   0x04, 0x08, 0x00, 0x40, 0x04, 0x00, 0x01, 0x28, 0x04, 0x08, 0x00, 0x40,
   0x04, 0x00, 0x01, 0x7c, 0x04, 0x08, 0x00, 0x40, 0x04, 0x00, 0x01, 0x44,
   0x04, 0x08, 0x00, 0x40, 0x04, 0x00, 0x01, 0x44, 0x04, 0x08, 0x00, 0x40,
   0x04, 0x00, 0x01, 0x44, 0x04, 0x08, 0x00, 0x40, 0x04, 0x00, 0x01, 0x44,
   0x04, 0x08, 0x00, 0x40, 0x08, 0x00, 0x01, 0x44, 0x04, 0x08, 0x00, 0x20,
   0x08, 0x00, 0x01, 0x44, 0x04, 0x08, 0x00, 0x20, 0x08, 0x00, 0x01, 0x44,
   0x04, 0x08, 0x00, 0x20, 0x08, 0x00, 0x01, 0x82, 0x04, 0x08, 0x00, 0x20,
   0x10, 0x00, 0x01, 0x82, 0x04, 0x08, 0x00, 0x10, 0x10, 0x00, 0x01, 0x82,
   0x04, 0x08, 0x00, 0x10, 0x10, 0x00, 0x7f, 0x82, 0xfc, 0xf9, 0x03, 0x10,
   0x20, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x08, 0x20, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x08, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x04,
   0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x02, 0x80, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x02, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01,
   0x00, 0x02, 0x00, 0x00, 0x00, 0x00, 0x80, 0x00, 0x00, 0x0c, 0x00, 0x00,
   0x00, 0x00, 0x60, 0x00, 0x00, 0x10, 0x00, 0x00, 0x00, 0x00, 0x10, 0x00,
   0x00, 0xe0, 0xff, 0xff, 0xff, 0xff, 0x0f, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
```

```
   0x10, 0x00, 0x41, 0x82, 0x20, 0x08, 0x02, 0x10, 0x10, 0x00, 0x41, 0x82,
   0x20, 0x08, 0x02, 0x10, 0x10, 0x00, 0x7f, 0xfe, 0x20, 0xf8, 0x03, 0x10,
   0x20, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x08, 0x20, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x08, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x04,
   0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x02, 0x80, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x02, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01,
   0x00, 0x02, 0x00, 0x00, 0x00, 0x00, 0x80, 0x00, 0x00, 0x0c, 0x00, 0x00,
   0x00, 0x00, 0x60, 0x00, 0x00, 0x10, 0x00, 0x00, 0x00, 0x00, 0x10, 0x00,
   0x00, 0xe0, 0xff, 0xff, 0xff, 0xff, 0x0f, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
```

```
   0x00, 0x00, 0x00, 0x10, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x10,
   0x00, 0x00, 0xc0, 0x00, 0x00, 0x00, 0x00, 0x10, 0x00, 0x00, 0x30, 0x00,
   0x00, 0x00, 0x00, 0x10, 0x00, 0x00, 0x08, 0x00, 0x00, 0x00, 0x00, 0x10,
   0x00, 0x00, 0x06, 0x00, 0x00, 0x00, 0x00, 0x10, 0x00, 0x00, 0x01, 0x00,
   0x00, 0x00, 0x00, 0x10, 0x00, 0xc0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x10,
   0x00, 0x30, 0x00, 0x00, 0x00, 0x00, 0x00, 0x10, 0x00, 0x08, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x10, 0x00, 0x06, 0x00, 0x00, 0x00, 0x00, 0x00, 0x10,
   0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x10, 0xc0, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x10, 0x30, 0xe0, 0x9f, 0x7f, 0x12, 0xc8, 0x3f, 0x10,
   0x08, 0x20, 0x90, 0x40, 0x12, 0x08, 0x04, 0x10, 0x08, 0x20, 0x90, 0x40,
   0x32, 0x08, 0x04, 0x10, 0x08, 0x20, 0x90, 0x40, 0x32, 0x08, 0x04, 0x10,
   0x08, 0x20, 0x90, 0x40, 0x32, 0x08, 0x04, 0x10, 0x08, 0x20, 0x90, 0x40,
   0x52, 0x08, 0x04, 0x10, 0x08, 0x20, 0x90, 0x40, 0x52, 0x08, 0x04, 0x10,
   0x08, 0x20, 0x90, 0x40, 0x52, 0x08, 0x04, 0x10, 0x08, 0x20, 0x90, 0x40,
   0x92, 0x08, 0x04, 0x10, 0x08, 0x20, 0x90, 0x40, 0x92, 0x08, 0x04, 0x10,
   0x08, 0xe0, 0x9f, 0x7f, 0x92, 0x08, 0x04, 0x10, 0x08, 0x20, 0x80, 0x01,
   0x12, 0x09, 0x04, 0x10, 0x08, 0x20, 0x80, 0x01, 0x12, 0x09, 0x04, 0x10,
   0x08, 0x20, 0x80, 0x02, 0x12, 0x09, 0x04, 0x10, 0x08, 0x20, 0x80, 0x02,
   0x12, 0x09, 0x04, 0x10, 0x08, 0x20, 0x80, 0x04, 0x12, 0x0a, 0x04, 0x10,
   0x08, 0x20, 0x80, 0x04, 0x12, 0x0a, 0x04, 0x10, 0x08, 0x20, 0x80, 0x08,
   0x12, 0x0a, 0x04, 0x10, 0x08, 0x20, 0x80, 0x10, 0x12, 0x0c, 0x04, 0x10,
   0x08, 0x20, 0x80, 0x10, 0x12, 0x0c, 0x04, 0x10, 0x08, 0x20, 0x80, 0x20,
   0x12, 0x0c, 0x04, 0x10, 0x08, 0x20, 0x80, 0x20, 0x12, 0x08, 0x04, 0x10,
   0x08, 0x20, 0x80, 0x40, 0x12, 0x08, 0x04, 0x10, 0x08, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x10, 0x08, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x10,
   0x08, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x10, 0x08, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x10, 0x08, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x10,
   0xf8, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x1f, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
```

```
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    position_file.c
*
*
* FILE FUNCTION:
*
*   This file contains the routines which maintain the Position directories and
*   process the popup frames which allow the user to select and create Position
*   directories.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /home/project/3531/Docu/GCB.spec.doc
*
*
* FILE MODULES:
*
*   build_cre_pos_popup()    - build the create Position popup
*   build_sel_pos_popup()    - build the select Position popup
*   cbr_cre_pos()            - display the create Position popup
*   cbr_new_position()       - process CREATE and CANCEL in create Position
*   cbr_pos_selected()       - user has selected a Position or directory
*   cbr_sel_pos()            - display the select Position popup, process CANCEL
*   justName()               - removes the .POS extension from a string
*   justPath()               - removes the Position name and extension from a str
*   load_pos_list()          - load the Position dirs and dir names into a list
*   valid_pos_name()         - check for a Position directory name
*
*****************************************<----->*****************************************/

#include <stdio.h>
#include <sys/file.h>
#include <sys/types.h>
#include <sys/time.h>
#include <dirent.h>

#include <X11/Intrinsic.h>
#include <Xm/Xm.h>

#include "gcb.h"
#include "cbr.h"
#include "widgets.h"
#include "position_file.h"
#include "element_file.h"
#include "constants.h"
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  build_cre_pos_popup()
*
* MODULE FUNCTION:
*
*   This routine builds the create Position popup.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  build_cre_pos_popup( parent )

    Widget  parent;

{
    int n;


    dlg_cre_pos = cr_popup( NULLS, parent, "Create Position" );

    FormW = cr_form( NULLS, dlg_cre_pos, NULL, NULL );
    set_attribs( FORM, FormW, 600, 225, XmRESIZE_NONE );

    /*
     *  Create the field labels, then the text entry fields, then the buttons.
     */

    lbl_cre_pos_ne    = cr_label(NULLS,FormW,"New Position Name:",0,15,25,9,IGNORE);
    lbl_cre_pos_ppath = cr_label(NULLS,FormW,"Position Path:",   0,30,40,13,IGNORE);
    cr_label(NULLS,FormW,"New Position Directory:",0,45,55,4, IGNORE);
    txt_cre_pos_nd    = cr_label(NULLS,FormW,"",0,45,55,32,IGNORE);

    txt_cre_pos_ne   = cr_text(NULLS,FormW,lbl_cre_pos_ne,   NULL,NULLS,FALSE,1,62);
    txt_cre_pos_ppath= cr_text(NULLS,FormW,lbl_cre_pos_ppath,NULL,NULLS,FALSE,1,62);

    btn_cre_pos_show = cr_command( NULLS,FormW,"Show New POS",cbr_text_proc,1);
    DoneW            = cr_command( NULLS,FormW,"Create",      cbr_new_position,DONE);
    CancelW          = cr_command( NULLS,FormW,"Cancel",      cbr_new_position,CANCEL);
    HelpW            = cr_command( NULLS,FormW,"Help",        cbr_help, CREATE_POS);

    cr_separator( NULLS, FormW, 70, IGNORE, 1, 99 );

    /*
     *  Set the attributes for the text fields and the buttons.
     */

    XtAddCallback( txt_cre_pos_ppath, XmNactivateCallback, cbr_text_proc, 1);
    XtAddCallback( txt_cre_pos_ne,    XmNactivateCallback, cbr_text_proc, 0);

    set_position( txt_cre_pos_ne,   15, IGNORE, 32, IGNORE);
    set_position( txt_cre_pos_ppath,30, IGNORE, 32, IGNORE);
    set_position( CancelW,          85, IGNORE,  5, IGNORE);
    set_position( btn_cre_pos_show, 85, IGNORE, 30, IGNORE);
    set_position( DoneW,            85, IGNORE, 55, IGNORE);
    set_position( HelpW,            85, IGNORE, 80, IGNORE);
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  build_sel_pos_popup()
*
* MODULE FUNCTION:
*
*   This routine builds the select Position popup.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  build_sel_pos_popup( parent )

    Widget  parent;

{
    dlg_sel_pos = cr_popup( NULLS,  parent, "Select Position" );
    FormW = cr_form ( NULLS, dlg_sel_pos, NULL, NULL );
    set_attribs( FORM, FormW, 500, 400, XmRESIZE_NONE );

    /*
     *  Create the labels and text fields which show the current position path
     *  and available positions.
     */

    cr_label(NULLS,FormW,"Position Path:",0, 5,10,4,  IGNORE);
    lbl_sel_pos_path  = cr_label(NULLS,FormW,"/home",          0, 5,10,30, IGNORE);
    cr_label( NULLS, FormW, "Position List:", 0,15,20,4,IGNORE );

    list_sel_pos = cr_list( NULLS, FormW, 14, 255 );
    XtAddCallback( list_sel_pos, XmNbrowseSelectionCallback, cbr_pos_selected, NULL );

    CancelW = cr_command( NULLS, FormW, "Cancel", cbr_sel_pos, CANCEL  );
    HelpW   = cr_command( NULLS, FormW, "Help",   cbr_help,    SELECT_POS );

    cr_separator( NULLS, FormW, 85, IGNORE, 1, 99 );

    set_position( XtParent(list_sel_pos),   15, IGNORE, 30, IGNORE);
    set_position( CancelW,                  93, IGNORE, 10, IGNORE);
    set_position( HelpW,                    93, IGNORE, 75, IGNORE);
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_cre_pos()
*
* MODULE FUNCTION:
*
*   This routine pops up the create Position popup.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_cre_pos( w, client_data, call_data )

    Widget  w;
    caddr_t client_data,
            call_data;

{
    if (Mode != EditSymbol)
        return;

    newName[0] = NULL;

    /*
     *  Use the existing position path if we have one, otherwise start at
     *  root.
     */

    if ( ValidPosition )
        {
        strcpy( newPath, PositionPath );
        justPath( newPath );
        XmTextSetString(txt_cre_pos_ppath, newPath);
        }
    else
        {
        strcpy( newPath, "/" );
        XmTextSetString(txt_cre_pos_ppath, "/");
        }

    XtManageChild( dlg_cre_pos );
    busy( dlg_cre_pos, FALSE );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_new_position()
*
* MODULE FUNCTION:
*
*   This routine processes the user button presses in the create Position
*   popup.  This routine handles both the CREATE and CANCEL buttons.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_new_position( w, client_data, call_data )

    Widget  w;
    caddr_t client_data,
            call_data;

{
    Arg         args[1];
    XmString    tcs;


    /*
     *  User wants to create a new Position directory.
     */

    if ( (int) client_data == DONE )
        {
        if ( mkdir(newPath,0777) )
            user_ack("Error: couldn't create new Position directory, check path");
        else
            {
            user_ack("Position directory created, use 'Select Position' to select");
            XtUnmanageChild( dlg_cre_pos );
            }
        }
    else

        /*
         *  User wants to abort from creating a new Position path, clear the
         *  text/label strings and then take down the popup.
         */

        if ( (int) client_data == CANCEL )
            {
            tcs = XmStringCreate( "", XmSTRING_DEFAULT_CHARSET );
            XtSetArg( args[0], XmNlabelString, tcs );
            XtSetValues( txt_cre_pos_nd, args, 1 );
            XmStringFree( tcs );

            XmTextSetString( txt_cre_pos_ne, NULLS );
            XtUnmanageChild( dlg_cre_pos );
            }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_pos_selected()
*
* MODULE FUNCTION:
*
*   This routine responds to the user's selecting a Position or new directory
*   from the list.  If the user selects a new directory, the current working
*   directory is changed and the list loaded with the new directorie's
*   contents.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_pos_selected( w, client_data, call_data )

    Widget                  w;
    caddr_t                 client_data;
    XmListCallbackStruct    *call_data;

{
    char    *ptr,
            *string;

    XmStringGetLtoR(call_data->item, XmSTRING_DEFAULT_CHARSET, &string);

    /*
     *  See if the entry is a directory, if it is move to the new directory
     *  and load the list again.
     */

    busy( dlg_sel_pos, TRUE );

    ptr = &string[strlen(string)-1];
    if ( *ptr == '/' )
        {
        chdir( string );
        load_pos_list();
        busy( dlg_sel_pos, FALSE );
        return;
        }

    /*
     *  The user has selected a Position directory.  First see if they
     *  need to save their work.
     */

    if ( SaveNeeded )
        save_curr_element();

    /*
     *  Now clear the work area and init the variables.
     */

    reinit_element_vars();
    init_comp_vars();

    /*
     *  Set the PositionPath to the new directory.
     */

    if ( chdir(string) )
        {
        user_ack("Error: setting to selected Position directory");

        /*
         *  Revert back to the previous Position directory.
         */

        if ( chdir(PositionPath) )
            {
            ValidPosition = FALSE;
            upd_pos_panel( COMP_PURPOSE );
            XtUnmanageChild( dlg_sel_pos );
            return;
            }

        /*
         *  Using the previous Position directory.
         */

        else
            {
            user_ack("Using previously selected Position directory");
            ValidPosition = TRUE;
            upd_pos_panel( COMP_PURPOSE );
            XtUnmanageChild( dlg_sel_pos );
            return;
            }
        }

    /*
     *  The current directory has been successfully set to the new Position
     *  directory.
     */

    load_curr_dir( PositionPath );

    strcpy( pPosition, string );
    pPosition[strlen(string)-4] = NULL;

    /*
     *  Update the status indicators.
     */

    ValidPosition = TRUE;
    upd_mode_panel();
    upd_pos_panel( COMP_PURPOSE );

    XtUnmanageChild( dlg_sel_pos );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_sel_pos()
*
* MODULE FUNCTION:
*
*   This routine pops up the select Position popup.  This routine is also called
*   by the CANCEL button in the popup.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_sel_pos( w, client_data, call_data )

    Widget  w;
    caddr_t client_data;
    caddr_t call_data;

{
    if (Mode != EditSymbol)
        return;

    /*
     *  User wants to abort from selecting a position, take down the popup.
     */

    if ( (int) client_data == CANCEL )
        {
        XtUnmanageChild( dlg_sel_pos );
        return;
        }

    /*
     *  Load and show the select Position popup.
     */

    if (! load_pos_list() )
        {
        XtManageChild( dlg_sel_pos );
        busy( dlg_sel_pos, FALSE );
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  justName()
*               justPath()
*
* MODULE FUNCTION:
*
*   The justName routine removes the .POS extension from a string.  If the
*   specified string does not end in the proper extension, the specified
*   string is not modified.  If the specified string contains the proper
*   extension, the "." which starts the extension is replaced with a NULL to
*   terminate the string.
*
*   The justPath routine removes the Position name and extension from a full
*   path name.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  justName( name )

    char *name;

{
    char *ptr;


    ptr = name;

    if ( *ptr == 'S' )
        {
        ptr--;
        if ( *ptr == 'O' )
            {
            ptr--;
            if ( *ptr == 'P' )
                {
                ptr--;
                if ( *ptr == '.' )
                    *ptr = NULL;
                }
            }
        }
}


void  justPath( path )

    char *path;

{
    char *ptr;


    /*
     *  Go out to the end of the string.
     */
```

```
    ptr = path;
    while ( *ptr != NULL )
        ptr++;
    ptr--;

    /*
     *  Now see if there is a Position extension, if so, remove the Position
     *  name and extension.
     */

    if ( *ptr == 'S' )
        {
        ptr--;
        if ( *ptr == 'O' )
            {
            ptr--;
            if ( *ptr == 'P' )
                {
                ptr--;
                if ( *ptr == '.' )
                    {
                    while ( *ptr != '/' )
                        {
                        *ptr = NULL;
                        ptr--;
                        }
                    }
                }
            }
        }

    /*
     *  Make sure the modified string ends in a '/'.
     */

    if ( *ptr != '/' )
        {
        ptr++;
        *ptr = '/';
        ptr++;
        *ptr = NULL;
        }

    return;
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  load_pos_list()
*
* MODULE FUNCTION:
*
*   This routine loads the Position names in the current directory into the
*   position list.  Subdirectories of the current working directory are also
*   entered into the list.  This will allow the user to traverse directories
*   within the file system.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  load_pos_list()
{
    Arg             args[1];
    DIR             *dir;
    XmString        tcs;
    struct dirent   *dp;
    int             cnt,
                    i;

    /*
     *  Get the current directory to display to the user in the select
     *  position popup.
     */

    load_curr_dir( pwd );

    tcs = XmStringCreate( pwd, XmSTRING_DEFAULT_CHARSET );
    XtSetArg( args[0], XmNlabelString, tcs );
    XtSetValues( lbl_sel_pos_path, args, 1 );
    XmStringFree( tcs );

    /*
     *  Delete all the list entries in the position selection list widget.
     */

    XmListDeleteAllItems( list_sel_pos );

    /*
     *  Open the current directory and read all valid entries: mainly directories.
     */

    if ((dir = opendir("./")) == NULL)
        {
        elog(1,"load_pos_list() - couldn't read directory");
        return( ERR );
        }

    for ( cnt=0, dp=readdir(dir); dp!=NULL; dp=readdir(dir) )
        {
        strcpy( selList[cnt], dp->d_name );
        if (! valid_pos_name(selList[cnt]) )
            cnt++;
        if ( cnt == MAX_FILES )
            {
```

```
        user_ack("Error: exhausted file list - notify developer");
        elog(1,"load_pos_list() - exhausted file list");
        return( ERR );
        }
    }

/*
 *  Sort the list and install the list into the selection widget.
 */

if ( cnt > 0 )
    {
    qsort( selList, cnt, MAX_NAME, strcmp );

    /*
     *  We want the lexigraphical order of these reversed.
     */

    if ((!strcmp(selList[0],"../")) &&
        (!strcmp(selList[1],"./")))
        {
        strcpy( selList[0], "./"  );
        strcpy( selList[1], "../" );
        }

    /*
     *  Install the strings into the select Position list widget.
     */

    for (i=0; i<cnt; i++)
        {
        tcs = XmStringCreate( selList[i], XmSTRING_DEFAULT_CHARSET );
        XmListAddItem( list_sel_pos, tcs, i+1 );
        XmStringFree( tcs );
        }

    }
    return( OK );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  valid_pos_name()
*
* MODULE FUNCTION:
*
*   This routine validates the position name.  This routine will append
*   a "/" to a directory name - the incoming string pointer must contain enough
*   allocated space at the end of the string to accommidate the appended
*   character.  This routine is used to prepare a string for entry into the
*   Select Position scrolled list.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  valid_pos_name( name )

    char *name;

{
    char        *ptr;
    int         len;
    struct stat stat_buf;


    len = strlen( name );

    /*
     *  Check for "." and "..".
     */

    if ((! strcmp(name,"." )) ||
        (! strcmp(name,".."  )) )
        {
        strcat( name, "/" );
        return( OK );
        }

    /*
     *  Check for ".POS" extension.
     */

    ptr = &name[len-1];
    if ( *ptr == 'S' )
        {
        ptr--;
        if ( *ptr == 'O' )
            {
            ptr--;
            if ( *ptr == 'P' )
                {
                ptr--;
                if ( *ptr == '.' )
                    return( OK );
                }
            }
        }
```

```
    /*
     *  Check for a directory.
     */

    stat(name,&stat_buf);
    if (stat_buf.st_mode & S_IFDIR)
        {
        strcat( name, "/" );
        return( OK );
        }

    return( ERR );
}
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    position_file.h
*
*
* FILE FUNCTION:
*
*   Variable declarations and function prototypes for the position_file.c
*   routines.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /home/project/3531/Docu/GCB.spec.doc
*
*
* FILE MODULES:
*
*   N/A
*
*****************************************<----->*****************************************/


char        newName[MAX_NAME],
            newPath[MAX_PATH],      /*  path to Position directory during create    */
            pwd[MAX_PATH];

/*
 *  Function prototypes.
 */

XtCallbackProc  cbr_new_position(),
                cbr_pos_selected(),
                cbr_sel_pos();

int             load_pos_list(),
                valid_pos_name();

void            justName(),
                justPath();
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    print.c
*
* FILE FUNCTION:
*
*   This file contains the routines which print the Element Work Area's graphical
*   symbols and lines.
*
*
* FILE MODULES:
*
*   build_print_elem_popup()     - select Element Work Area print attributes/options
*   cbr_print()                  - manages the print Element popup, called from menus
*   cbr_print_comp()             - determines print attributes and makes calls to print
*   copy_ps_template()           - copies template file into current working directory
*   draw_log_box()               - draws each of text fields for if/set symbol
*   draw_lower_box()             - in comp report, draws whole element in reduced mode
*   draw_upper_box()             - in comp report, draws box with status fields, purpose
*   make_ps_file()               - makes calls to build a PostScript disk file
*   make_ps_file_ifset()         - prints page for each if/set, with text fields
*   print_page()                 - sets up the image clip region
*
*****************************************<----->*****************************************/

#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>

#include "gcb.h"
#include "cbr.h"
#include "widgets.h"
#include "menu.h"
#include "lines.h"
#include "constants.h"
#include "fonts.h"
#include "print.h"
#include "element_file.h"
#include "symbol.h"

#define ULCY     1850
#define URCX     1400
#define LRCY     1350
#define BOPY     100
#define ELEM_LIST_TOP    1700

#define ULCY_log     1550
#define LRCY_log     1150

#define ULCY_expr    1050
#define LRCY_expr    650

#define ULCY_comment     550
#define LRCY_comment     150
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  build_print_elem_popup()
*
* MODULE FUNCTION:
*
*   This routine builds the popup which allows the user to select several attributes
*   during the printing of the element work area.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  build_print_elem_popup( parent )

    Widget parent;

{
    dlg_print_elem = cr_popup( NULLS, parent, "Print Element" );

    FormW = cr_form( NULLS, dlg_print_elem, NULL, NULL);
    set_attribs( FORM, FormW, 450, 200, XmRESIZE_NONE );

    cr_label( NULLS, FormW, "Print Scale:", 0, 15, IGNORE, 9, IGNORE );

    rb_print_elem = cr_radio_box( NULLS, FormW, XmHORIZONTAL );

    tgl_print_elem_norm = cr_toggle( NULLS, rb_print_elem, "Normal", NULL, 0, 0);
    tgl_print_elem_red  = cr_toggle( NULLS, rb_print_elem, "Reduced",NULL, 0, 0);
    arm_tgl( tgl_print_elem_norm );

    cr_label( NULLS, FormW, "Number of Copies: ", 0,35,IGNORE,9,IGNORE );
    txt_print_elem_nc = cr_text ( NULLS, FormW, NULL, NULL, "1", FALSE, 1, 2);

    cr_label( NULLS, FormW, "Printer Name: ", 0,55,IGNORE,9,IGNORE );
    txt_print_elem_pname = cr_text ( NULLS, FormW, NULL, NULL, "ps", FALSE, 1, 8);

    cr_separator( NULLS, FormW, 75, IGNORE, 1, 99 );

    DoneW   = cr_command( NULLS, FormW, "Print",  cbr_print_comp, CBR_COMP_TYPE_DONE);
    CancelW = cr_command( NULLS, FormW, "Cancel", cbr_print_comp, CBR_COMP_TYPE_CANCEL);
    HelpW   = cr_command( NULLS, FormW, "Help",   cbr_help,       PRINT_ELEM);

    set_position( rb_print_elem,        13, IGNORE, 40, IGNORE );
    set_position( txt_print_elem_pname, 53, IGNORE, 40, IGNORE );
    set_position( txt_print_elem_nc,    35, IGNORE, 40, IGNORE);
    set_position( CancelW,              85, IGNORE,  2, IGNORE);
    set_position( DoneW,                85, IGNORE, 40, IGNORE);
    set_position( HelpW,                85, IGNORE, 78, IGNORE);
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_print()
*
* MODULE FUNCTION:
*
*   This routine processes calls from the print menu buttons.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_print( w, client_data, call_data )

    Widget  w;
    int     client_data;
    caddr_t call_data;

{
    FILE                    *fp, *copy_ps_template();
    struct symbol_entry     *templist;
    char                    *temp_elemname[MAX_NAME];

    /*
     *  Can't print without a Valid Element.
     */

    if (! ValidElement )
        {
        error_handler( NO_ELEMENT, NULL );
        return;
        }

   if ( client_data == PRINT_REP )
        {

    /*
     *  we are not printing a report; pop up ordinary print menu.
     */

        Report = 0;
        XtManageChild( dlg_print_elem );
        return;
        }
   else if ( client_data == PRINT_REP1 )
        {

        /*
         *  tell print routines that we are printing a report,
         *  not just a page.
         */

        Report = 1;

        /*
         *  copy ps_template to ps_file; open it for appending.
         */

        fp = (FILE *)copy_ps_template();

        PrintScale = REDUCED;
        Copies = 1;

        /*
         *  print the title page with comp purpose
         */

        make_ps_file_title( fp );

        /*
         *  print the list of comp elements and global
         *  variables. Second parameter is non_zero to indicate that we are
         *  looking only for global variables.
         */

        make_ps_file_elemlist( fp, 1, NULL );

        /*
         *  save the name of the current element to restore after print.
         */

        strcpy( temp_elemname, ElementFile );

        ElementListType = ELEMENT;

        /*
         *  go through list of elements in symbol table for this comp;
         *  for each, print the element and its if/set symbols.
         */

        templist = (struct symbol_entry *)symbol_table;
        while ( templist )
            {
            if ( templist->se_type & PROCEDURE )
                {

                /*
                 *  it's an element only if it isn't an intrinsic procedure like
                 *  sin, cos, etc. Also print only those elements that are used
                 *  in this comp.
                 */

                if ( ((! (templist->se_type & INTRINSIC)) && templist->se_use_count) ||
                     (! strcmp(RootElement,templist->se_symbol)) )
                    {

                    /*
                     *  Set this element to be the current element: copy this name
                     *  into GElementFile and ElementFile.
                     */

                    reinit_element_vars();

                    strcpy( ElementFile, templist->se_symbol );
                    strcpy( GElementFile, ElementFile );
                    strcat( GElementFile, EL_EXT );

                    /*
                     *  try to access this element; if unsuccessful, go to next
                     *  element.
                     */

                    if (! access(GElementFile,R_OK) )
```

```
                {

                /*
                 *  read this element into memory and the Symbol_Map.
                 *  if we can't, skip printing this element.
                 */

                if ( read_element_file() != ERR )
                    {

                    /*
                     *  print the element in reduced mode.
                     */

                    PrintScale = REDUCED;
                    make_ps_file( fp, Default );

                    /*
                     *  print the variables local to this element.
                     *  Second parameter is zero to show we are looking
                     *  for locals in the third parameter.
                     */

                    make_ps_file_elemlist( fp, 0, templist->se_symbol );

                    /*
                     *  for each IF/SET, draw the symbol top center, then the
                     *  3 boxes with text.
                     */

                    make_ps_file_ifset( fp );
                    }
                }
            }
        templist = templist->se_next;
        }

fclose( fp );

/*
 *  Make "system" call to lpr to print the PostScript file.
 */

elog(3,"make ps file ifset: result of lpr: %d", system("lpr -h ps_file") );

/*
 *  restore the name of the current element to its original value.
 */

reinit_element_vars();
strcpy( ElementFile, temp_elemname );

/*
 *  restore the current element to its original value.
 */

strcpy( GElementFile, ElementFile );
strcat( GElementFile, EL_EXT );

if ( access(GElementFile,R_OK) )
    {
    user_ack("can't access element file!");
    exit( ERR );
    }

if ( read_element_file() == ERR )
    {
    user_ack("can't restore element file!");
    exit( ERR );
    }

/*
 *  restore this symbol's (in)complete status.
 */

if ( complete(0, LINES_AND_EXPR) == ERR )
    upd_status( 1 );
else upd_status( 0 );

/*
 *  restore the status fields
 */

 upd_pos_panel( NO_CHANGE );
}
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_print_comp()
*
* MODULE FUNCTION:
*
*   This routine determines the attributes to use when printing (normal, reduced,
*   number of copies, etc.) and then makes the calls to build the PostScript disk
*   file.  Then a system call is made to print the PostScript file.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_print_comp( w, client_data, call_data )

    Widget  w;
    int     client_data;
    caddr_t call_data;

{
    FILE    *fp, *copy_ps_template();
    char    *sys_call[128];

    /*
     *  erase the bounded print box.
     */

    draw_ghost( FALSE, BStartx, BStarty, 0, 0 );

    /*
     *  Determine which button the user pressed in the popup.
     */

    switch ( client_data )
        {
        case CBR_COMP_TYPE_CANCEL :

            /*
             *  reset global vars, pop down print menu.
             */

            old_lrcx = -1;
            old_lrcy = -1;
            old_ulcx = 0;
            old_ulcy = 0;
            XtUnmanageChild( dlg_print_elem );
            return;

        /*
         *  The user wants to print, determine if they want to print normal or
         *  reduced, and then determine the number of copies to print.
         */

        case CBR_COMP_TYPE_DONE :
            if ( XmToggleButtonGetState(tgl_print_elem_norm) )
                PrintScale = NORMAL;
            else
                PrintScale = REDUCED;
            Copies = atoi( XmTextGetString(txt_print_elem_nc) );
            break;
        }

    busy( dlg_print_elem, TRUE );

    /*
     *  If we are zoomed out, zoom back in so as to restore the correct
     *  sizes of the symbols and lines.
     */

    if ( Zoomed )
        zoom( 1 );

    /*
     *  copy ps_template to ps_file; open it for appending.
     */

    if ( (fp = (FILE *)copy_ps_template()) == NULL )
        {
        user_ack("copy ps_template failed");
        exit( ERR );
        }

    PageNum = 0;

    /*
     *  old_lrcx is set only in the draw print box mode; if it is not 0,
     *  we have drawn a printing bounding box.  Generate a PostScript file
     *  on disk for printing.
     */

    if ( old_lrcx >=0 )
        {
        elog(3,"cbr_print_comp: making bounded box");
        make_ps_file( fp, Banded );
        }
    else
        {
        elog(3,"cbr_print_comp: using default box");
        make_ps_file( fp, Default );
        }

    /*
     *  close the file before printing.
     */

    fclose( fp );

    /*
     *  Make "system" call to lpr to print the PostScript file.
     */

           Copies = atoi( XmTextGetString(txt_print_elem_nc) );
    strcpy( sys_call, "lpr -h -P" );
    strcat( sys_call, XmTextGetString(txt_print_elem_pname) );
    strcat( sys_call, " ps_file" );
    elog(3,"cbr_print_comp: result of lpr: %d", system(sys_call) );

    /*
     *  Restore defaults.
     */

    old_lrcx = -1;
```

```
    old_lrcy = -1;
    old_ulcx = 0;
    old_ulcy = 0;

    /*
     *  Take down the popup.
     */

    busy( dlg_print_elem, FALSE );
    XtUnmanageChild( dlg_print_elem );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  copy_ps_template()
*
* MODULE FUNCTION:
*
*   This routine  copies the template file into the current working directory
*   for appending.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

FILE  *copy_ps_template()

{
    FILE    *fp;
    char    sys_call[256];

    /*
     *  create and execute a cp system call
     */

    strcpy( sys_call, "cp " );
    strcat( sys_call, Swd   );
    strcat( sys_call, "/ps_template ps_file" );

    if ( system(sys_call) )
        elog(1,"make_ps_file: system cp call returned nonzero");

    if (! (fp = fopen("ps_file","a")) )
        {
        user_ack("Unable to open the PostScript file for appending");
        elog(1,"make_ps_file: Unable to open the PostScript file for appending");
        return( NULL );
        }

    return( (FILE *)fp );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  draw_log_box()
*
* MODULE FUNCTION:
*
*   This routine draws one of the logical/expr/comment boxes for each if/set
*   symbol in the element.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  draw_log_box( fp, sym_num, box_type )

    FILE    *fp;
    int     sym_num, box_type;

{
    char    result[MAX_PURPOSE];
    int     ulcy;

    PrintScale = REDUCED;
    fprintf( fp, "0 SetScale\n");
    fprintf( fp, "newpath\n");

    /*
     *  convert proper if/set field to postscript-ready form
     */

    result[0] = '\0';
    switch ( box_type )
        {
        case 0: if ( Symbol_Map[sym_num].Sym.IfSym.logical_expr )
                    cvt_to_str_array(Symbol_Map[sym_num].Sym.IfSym.logical_expr,result);
                ulcy = ULCY_log;
                break;
        case 1: if ( Symbol_Map[sym_num].Sym.IfSym.comp_expr )
                    cvt_to_str_array( Symbol_Map[sym_num].Sym.IfSym.comp_expr, result );
                ulcy = ULCY_expr;
                break;
        case 2: if ( Symbol_Map[sym_num].Sym.IfSym.comment )
                    cvt_to_str_array( Symbol_Map[sym_num].Sym.IfSym.comment, result );
                ulcy = ULCY_comment;
                break;
        }

    if ( result[0] == '\0' )

        /*
         *  required text field is missing; print blank
         */

        strcpy( result, "[()]" );

    fprintf( fp, "%d %d %d %d %s %d PrintReportBox\n",
        pix_to_ps(pix_left_margin()+100),
        pix_to_ps(ulcy), pix_to_ps(ULCY_log-LRCY_log),
        pix_to_ps(URCX - (pix_left_margin()+100)),  result, box_type );

    PrintScale = NORMAL;
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  draw_lower_box()
*
* MODULE FUNCTION:
*
*   This routine draws a box around the  entire element in reduced mode as
*   part of the report.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  draw_lower_box( fp, lrcx, lrcy )

    FILE    *fp;
    int     lrcx, lrcy;

{
    fprintf( fp, "newpath\n");
    fprintf( fp, "%d %d %d %d drawBox\n",  pix_to_ps(pix_left_margin()),
        pix_to_ps(pix_centering_height()), pix_to_ps(lrcx - pix_left_margin()),
         pix_to_ps(pix_centering_height() - (pix_centering_height()-lrcy)) );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  draw_upper_box()
*
* MODULE FUNCTION:
*
*   This routine draws the upper box of the first page of an RMS report, including
*   the element, comp, and position names, dates, and element type.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  draw_upper_box( fp )

    FILE    *fp;

{
    char        result[MAX_PURPOSE];
    XmString    tcs;
    char        *complete;
    Arg         args[1];
    int         i;

    fprintf( fp, "newpath\n");
    fprintf( fp, "%d %d %d %d drawBox\n",  pix_to_ps(pix_left_margin()), pix_to_ps(ULCY),
         pix_to_ps(ULCY - LRCY), pix_to_ps(URCX - pix_left_margin()) );

    fprintf( fp, "%d SetScale\n", 0 );

    /*
     *  draw element name, author, date created, position, and comp.
     */

    fprintf( fp, "%d %d moveto\n",  pix_to_ps(pix_left_margin() + 20), pix_to_ps(1770) );
    fprintf( fp, "/Courier-Bold findfont 12 scalefont setfont\n");
    fprintf( fp, "(Position:) show\n" );
    fprintf( fp, "%d %d moveto\n",  pix_to_ps(pix_left_margin() + 280), pix_to_ps(1770));
    fprintf( fp, "%d SetScale\n", 0 );
    fprintf( fp, "(%s) show\n", pPosition );

    fprintf( fp, "%d %d moveto\n",  pix_to_ps(pix_left_margin() + 20), pix_to_ps(1730) );
    fprintf( fp, "/Courier-Bold findfont 12 scalefont setfont\n");
    fprintf( fp, "(Comp:) show\n" );
    fprintf( fp, "%d %d moveto\n",  pix_to_ps(pix_left_margin() + 280), pix_to_ps(1730));
    fprintf( fp, "%d SetScale\n", 0 );
    fprintf( fp, "(%s) show\n", CompFile );

    fprintf( fp, "%d %d moveto\n",  pix_to_ps(pix_left_margin() + 20), pix_to_ps(1690) );
    fprintf( fp, "/Courier-Bold findfont 12 scalefont setfont\n");
    fprintf( fp, "(Element Name:) show\n");
    fprintf( fp, "%d %d moveto\n",  pix_to_ps(pix_left_margin() + 280), pix_to_ps(1690));
    fprintf( fp, "%d SetScale\n", 0 );
    fprintf( fp, "(%s) show\n", ElementFile );

    fprintf( fp, "%d %d moveto\n",  pix_to_ps(pix_left_margin() + 20), pix_to_ps(1650) );
    fprintf( fp, "/Courier-Bold findfont 12 scalefont setfont\n");
    fprintf( fp, "(Element Type:) show\n");
    fprintf( fp, "%d %d moveto\n",  pix_to_ps(pix_left_margin() + 280), pix_to_ps(1650));
```

```
    fprintf( fp, "%d SetScale\n", 0 );
    if ( ElementType == LIB )
        fprintf( fp, "(LIBRARY) show\n" );
    else
        fprintf( fp, "(ELEMENT) show\n" );

    fprintf( fp, "%d %d moveto\n",  pix_to_ps(pix_left_margin() + 20), pix_to_ps(1610) )
;
    fprintf( fp, "/Courier-Bold findfont 12 scalefont setfont\n");
    fprintf( fp, "(Author:) show\n" );
    fprintf( fp, "%d %d moveto\n",  pix_to_ps(pix_left_margin() + 280), pix_to_ps(1610))
;
    fprintf( fp, "%d SetScale\n", 0 );
    fprintf( fp, "(%s) show\n", Author );

    fprintf( fp, "%d %d moveto\n",  pix_to_ps(pix_left_margin() + 20), pix_to_ps(1570) )
;
    fprintf( fp, "/Courier-Bold findfont 12 scalefont setfont\n");
    fprintf( fp, "(Created:) show\n" );
    fprintf( fp, "%d %d moveto\n",  pix_to_ps(pix_left_margin() + 280), pix_to_ps(1570))
;   fprintf( fp, "%d SetScale\n", 0 );
    fprintf( fp, "(%s) show\n", CreateDate );

    fprintf( fp, "%d %d moveto\n",  pix_to_ps(pix_left_margin() + 20), pix_to_ps(1530) )
;
    fprintf( fp, "/Courier-Bold findfont 12 scalefont setfont\n");
    fprintf( fp, "(Last Update:) show\n" );
    fprintf( fp, "%d %d moveto\n",  pix_to_ps(pix_left_margin() + 280), pix_to_ps(1530))
;
    fprintf( fp, "%d SetScale\n", 0 );
    fprintf( fp, "(%s) show\n", UpdateDate );

    fprintf( fp, "%d %d moveto\n",  pix_to_ps(pix_left_margin() + 20), pix_to_ps(1490) )
;
    fprintf( fp, "/Courier-Bold findfont 12 scalefont setfont\n");
    fprintf( fp, "(Status:) show\n" );
    fprintf( fp, "%d %d moveto\n",  pix_to_ps(pix_left_margin() + 280), pix_to_ps(1490))
;
    fprintf( fp, "%d SetScale\n", 0 );

    /*
     *  if status string = "Complete", print complete on report, else print
     *  incomplete.
     */

    XtSetArg( args[0], XmNlabelString, &tcs );
    XtGetValues( txt_status, args, 1 );
    XmStringGetLtoR( tcs, XmSTRING_DEFAULT_CHARSET, &complete );
    if ( !strcmp("Complete", complete) )
        fprintf( fp, "(Complete) show\n" );
    else
        fprintf( fp, "(Incomplete)  show\n" );

    /*
     *  print element purpose
     */

    cvt_to_str_array( ElementPurpose, result );
    fprintf( fp, "%d %d %d %s PrintElemPurpose\n", pix_to_ps(pix_left_margin() + 600),
        pix_to_ps(1770), pix_to_ps(1000), result );

    /*
     *  Set the scale: the font size.
     */

    if ( PrintScale == REDUCED )
        fprintf( fp, "%d SetScale\n", 2 );
    else
        fprintf( fp, "%d SetScale\n", 1 );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  make_elemlist_heading()
*
* MODULE FUNCTION:
*
*   This routine draws the labels over the list of elements on the title page.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  make_elemlist_heading( fp )

    FILE    *fp;

{
    /*
     *  move to top of page; print labels.
     */

    fprintf( fp, "%d %d moveto\n", pix_to_ps(pix_left_margin() + 100),
        pix_to_ps(ELEM_LIST_TOP) );
    fprintf( fp, "(Elements) show\n" );
    fprintf( fp, "%d %d moveto\n", pix_to_ps(pix_left_margin() + 100),
        pix_to_ps(ELEM_LIST_TOP) );
    fprintf( fp, "(________) show\n" );

    fprintf( fp, "%d %d moveto\n", pix_to_ps(pix_left_margin() + 500),
        pix_to_ps(ELEM_LIST_TOP) );
    fprintf( fp, "(Installed) show\n" );
    fprintf( fp, "%d %d moveto\n", pix_to_ps(pix_left_margin() + 500),
        pix_to_ps(ELEM_LIST_TOP) );
    fprintf( fp, "(_________) show\n" );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  make_elemlist_listing()
*
* MODULE FUNCTION:
*
*   This routine  draws the actual list of elements under the labels.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  make_elemlist_listing( fp, i )

    FILE    *fp;
    int     i;

{
    struct symbol_entry     *templist;
    char                    pn[3];

    /*
     *  go through symbol table; print each element name
     */

    templist = (struct symbol_entry *)symbol_table;
    while( templist )
        {
        if ( templist->se_type & PROCEDURE )
            {
            /*
             *  it's an element only if it isnt an intrinsic procedure like
             *  sin, cos, etc.  Also, print only those elements that are called.
             */

            if ( (! (templist->se_type & INTRINSIC)) && templist->se_use_count )
                {

                /*
                 *  if the vertical offset is too large for the current page,
                 *  print the current page, print the element list heading on
                 *  the next page, and reset the vertical offset.
                 */

                if ( i > 24 )
                    {

                    /*
                     *  convert the current page number to a string, print it.
                     *  then draw the current page.
                     */

                    itoa( PageNum, pn );
                    fprintf( fp, "(%s) EndPage\n", pn );
                    PageNum++;

                    /*
                     *  print heading on new page
                     */
```

```
                make_elemlist_heading( fp );

                /*
                 *  reset vertical offset
                 */

                i = 1;
                }

            /*
             *  print whether or not the element is installed
             */

            fprintf( fp, "%d %d moveto\n", pix_to_ps(pix_left_margin() + 100),
                pix_to_ps(ELEM_LIST_TOP) - (i*20) );
            fprintf( fp, "(%s) show\n", templist->se_symbol );
            fprintf( fp, "%d %d moveto\n", pix_to_ps(pix_left_margin() + 500),
                pix_to_ps(ELEM_LIST_TOP) - (i*20) );
            if ( templist->se_type & INSTALLED )
                fprintf( fp, "(yes) show\n" );
            else
                fprintf( fp, "(no) show\n" );
            i++;
            }
        }
    templist = templist->se_next;
    }

/*
 *  return vertical offset so far.
 */

return( i );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  make_varlist_heading()
*
* MODULE FUNCTION:
*
*   This routine draws the labels over the list of global variables.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  make_varlist_heading( fp, i, global )

    FILE    *fp;
    int     i, global;

    /*
     *  i is a vertical offset into the page; put varlist heading below this point.
     */

{
    /*
     *  move past bottom of element list; print headings for list of
     *  global variables.
     */

    fprintf( fp, "%d %d moveto\n", pix_to_ps(pix_left_margin() + 100),
        pix_to_ps(ELEM_LIST_TOP) - (i*20) );
    if ( global )
        {
        fprintf( fp, "(Global Variables) show\n" );
        fprintf( fp, "%d %d moveto\n", pix_to_ps(pix_left_margin() + 100),
            pix_to_ps(ELEM_LIST_TOP) - (i*20) );
        fprintf( fp, "(________________) show\n" );
        }
    else
        {
        fprintf( fp, "(Local Variables) show\n" );
        fprintf( fp, "%d %d moveto\n", pix_to_ps(pix_left_margin() + 100),
            pix_to_ps(ELEM_LIST_TOP) - (i*20) );
        fprintf( fp, "(________________) show\n" );
        }

    fprintf( fp, "%d %d moveto\n", pix_to_ps(pix_left_margin() + 500),
        pix_to_ps(ELEM_LIST_TOP) - (i*20) );
    fprintf( fp, "(Type) show\n" );
    fprintf( fp, "%d %d moveto\n", pix_to_ps(pix_left_margin() + 500),
        pix_to_ps(ELEM_LIST_TOP) - (i*20) );
    fprintf( fp, "(____) show\n" );

    fprintf( fp, "%d %d moveto\n", pix_to_ps(pix_left_margin() + 700),
        pix_to_ps(ELEM_LIST_TOP) - (i*20) );
    fprintf( fp, "(Dimensions) show\n" );
    fprintf( fp, "%d %d moveto\n", pix_to_ps(pix_left_margin() + 700),
        pix_to_ps(ELEM_LIST_TOP) - (i*20) );
    fprintf( fp, "(__________) show\n" );

    fprintf( fp, "%d %d moveto\n", pix_to_ps(pix_left_margin() + 980),
```

```
    pix_to_ps(ELEM_LIST_TOP) - (1*20) );
fprintf( fp, "(Use Count) show\n" );
fprintf( fp, "%d %d moveto\n", pix_to_ps(pix_left_margin() + 980),
    pix_to_ps(ELEM_LIST_TOP) - (1*20) );
fprintf( fp, "(_________) show\n" );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  make_ps_file()
*
* MODULE FUNCTION:
*
*   This routine creates a PostScript file based on Symbol_Map contents.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  make_ps_file( fp, mode )

    FILE    *fp;
    int mode;

{
    /*
     *  If we are printing the whole element (not using a banded-box), then print
     *  using the proper scale (normal or reduced print).
     */

    if ( mode == Default )
        {

        /*
         *  Print logical screen on 4 pages in 4 parts if scale is normal.
         */

        if ( PrintScale == NORMAL )
            {

            /*
             *  draw upper box with status and purpose info.
             */

            print_page( fp, 830, pix_centering_height(), 0,   0   );
            print_page( fp, 830, pix_centering_height(), 550, 0   );
            print_page( fp, 830, pix_centering_height(), 0,   600 );
            print_page( fp, 830, pix_centering_height(), 550, 600 );
            }

        /*
         *  print whole logical screen in 1 part if scale is reduced,
         */

        else
            {

            /*
             *  draw upper box with status and purpose info.
             */

            draw_upper_box( fp );

            /*
             *  draw lower box that will enclose the element drawing.
             */
```

```
        draw_lower_box( fp, 1400, 1200 );

        /*
         *  draw element.
         */

        print_page( fp, 1400, 1200, 0, 0 );
        }
    }

/*
 *  User wants to print only a portion of the element which was selected
 *  using a banded-box.
 */

else if ( mode == Banded )
    {

    /*
     *  Print the contents of the banded box.  In order for the banded box
     *  to appear in the ulc of the page, the ulc of the box must be 0, and
     *  the offset of the symbols must be the ulc of the box.
     */

    print_page( fp, abs(BStartx-old_lrcx), abs(BStarty-old_lrcy), BStartx,BStarty);
    }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  make_ps_file_elemlist()
*
* MODULE FUNCTION:
*
*   This routine creates a PostScript file based on Symbol_Map contents.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  make_ps_file_elemlist( fp, global, element_name )

    FILE    *fp;
    int     global;
    char    *element_name;

{
    struct symbol_entry     *templist = NULL, *varlist, *lookup_global_varlist();
    int                     i = 1, j;
    char                    pn[3];

    if ( global )

    /*
     *  Get list of global variables from symbol table.
     */

        templist = (struct symbol_entry *)lookup_global_varlist();
    else

    /*
     *  Get list of local variables for this element from symbol table.
     */

        templist = (struct symbol_entry *) lookup_local_varlist( element_name );

    if ( templist == NULL )

        /*
         *  there are no elements or local variables to list; return.
         */

         return;

    /*
     *  Set large font size
     */

    fprintf( fp, "newpath\n");
    fprintf( fp, "0 SetScale\n" );

    /*
     *  Draw border around the page.
     */

    fprintf( fp, "%d %d %d %d drawBox\n",  pix_to_ps(pix_left_margin()), pix_to_ps(ULCY),
         pix_to_ps(ULCY - BOPY), pix_to_ps(URCX - pix_left_margin()) );
```

```
/*
 *  print header for element list
 */

if ( global )
    {

    make_elemlist_heading( fp );

    /*
     *  print element list
     */

    i = make_elemlist_listing( fp, 1 );

    i++;
    }

/*
 *  if there are variables to list, print header for variable list,
 *  last parameter is non-zero if global variable heading will be
 *  printed, 0 for local.
 */

make_varlist_heading( fp, i, global );

i++;

while( templist )
    {
    if ( templist->se_type & VARIABLE )
        {

        /*
         *  if we're looking for globals and the variable is not a local
         *  or if we are looking only for locals, print the variable.
         */

        if ( ((global) && (! (templist->se_type & LOCAL_VAR))) ||
            (!global) )

            {

            /*
             *  if the vertical offset into the page (i) is too large,
             *  print this page, print the variable list header on the
             *  next page, and reset the vertical offset.
             */

            if ( i > 24 )
                {

                /*
                 *  convert the current page number to a string, print it.
                 *  then draw the current page.
                 */

                itoa( PageNum, pn );
                fprintf( fp, "(%s) EndPage\n", pn );
                PageNum++;

                /*
                 *  reset vertical offset
                 */

                i = 1;

                /*
                 *  print heading on new page
                 */

                make_varlist_heading( fp, i, global );

                i++;
                }

            /*
             *  print the variable name
             */

            fprintf( fp, "%d %d moveto\n", pix_to_ps(pix_left_margin() + 100),
                pix_to_ps(ELEM_LIST_TOP) - (i*20) );
            fprintf( fp, "(%s) show\n", templist->se_symbol );

            /*
             *  print the variable type
             */

            fprintf( fp, "%d %d moveto\n", pix_to_ps(pix_left_margin() + 500),
                pix_to_ps(ELEM_LIST_TOP) - (i*20) );

            if ( templist->se_type  & INTEGER )
                fprintf( fp, "(int) show\n" );
            else if ( templist->se_type  & FLOAT )
                fprintf( fp, "(float) show\n" );
            else if ( templist->se_type  & CHAR )
                fprintf( fp, "(string) show\n" );
            else if ( templist->se_type  & UNSIGNED )
                fprintf( fp, "(unsigned) show\n" );
            else if ( templist->se_type  & SHORT )
                fprintf( fp, "(short) show\n" );
            else
                fprintf( fp, "(double) show\n" );

            /*
             *  print the variable dimensions
             */

            fprintf( fp, "%d %d moveto\n", pix_to_ps(pix_left_margin() + 700),
                pix_to_ps(ELEM_LIST_TOP) - (i*20) );

            if (! (templist->se_type  & MATRIX) )
                fprintf( fp, "(1) show\n" );
            else
                for ( j=0; j<templist->se_num_dimensions; j++ )
                    {
                    fprintf( fp, "(%i) show\n", templist->se_subs[j] );
                    if ( (templist->se_num_dimensions-j) > 1 )
                        fprintf( fp, "( x ) show\n" );
                    }

            fprintf( fp, "%d %d moveto\n", pix_to_ps(pix_left_margin() + 980),
                pix_to_ps(ELEM_LIST_TOP) - (i*20) );

            /*
             *  print the variable use count
             */
```

```
            fprintf( fp, "(%i) show\n", templist->se_use_count );

            i++;
            }
        }
    templist = templist->se_next;
    }

/*
 *  deallocate global variable list.
 */

destroy_global_varlist( templist );
fprintf( fp, "stroke\n" );

/*
 *  convert the current page number to a string and print it
 *  then print the current page.
 */

itoa( PageNum, pn );
fprintf( fp, "(%s) EndPage\n", pn );
PageNum++;
}
```

```
/*****************************************<----->*************************************
*
* MODULE NAME:  make_ps_file_ifset()
*
* MODULE FUNCTION:
*
*   This routine creates a PostScript file based on Symbol_Map contents.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*************************************/

int  make_ps_file_ifset( fp )

    FILE    *fp;

{
    int     pswidth, psheight, psulcx, psulcy, i;
    char    result[300], pn[3];

    fprintf( fp, "newpath\n");

    /*
     *  pix_to_ps depends on the PrintScale, so set it to NORMAL.
     */

    PrintScale = NORMAL;

    /*
     *  for each if/set, draw the symbol and associated text boxes on a
     *  separate page.
     */

    for ( i=0; i<MAX_SYMBOLS; i++ )
        if ( (Symbol_Map[i].symbol_type == IF) ||
            (Symbol_Map[i].symbol_type == SET) )
            {

            /*
             *  place the symbol at top middle of page.
             */

            PrintScale = REDUCED;
            psulcx = pix_to_ps( (pix_left_margin() + URCX) / 2 - Symbol_Map[i].width );
            psulcy = pix_to_ps( ULCY );
            PrintScale = NORMAL;

            /*
             *  determine symbol's height and width in postscript units.
             */

            pswidth = pix_to_ps( Symbol_Map[i].width );
            psheight = pix_to_ps( Symbol_Map[i].height );

            if ( Symbol_Map[i].font == small_font )
                fprintf( fp, "%d SetScale\n", 1 );
            else
                fprintf( fp, "%d SetScale\n", 0 );
```

```
        /*
         *  print the logical expression if there is one, else print the
         *  comp expr.
         */

        if ( Symbol_Map[i].Sym.IfSym.logical_expr )
            cvt_to_str_array( Symbol_Map[i].Sym.IfSym.logical_expr, result );
        else
            cvt_to_str_array( Symbol_Map[i].Sym.IfSym.comp_expr, result );

        if ( Symbol_Map[i].symbol_type == IF )
          .   fprintf( fp, "%d %d %d %d %d %s iif\n",
                psulcx, psulcy, pswidth, psheight, pix_to_ps(23), result );
        else
             fprintf( fp, "%d %d %d %d %s set\n", psulcx, psulcy,
                psheight, pswidth,  result);

        /*
         *  draw the logical, expression, and comment boxes and the
         *  associated text for this symbol.
         */

        draw_log_box( fp, i, 0 );
        draw_log_box( fp, i, 1 );
        draw_log_box( fp, i, 2 );

        /*
         *  convert the current page number to a string and print it
         *  draw this page, get ready for next page.
         */

        itoa( PageNum, pn );
        fprintf( fp, "(%s) EndPage\n", pn );
        PageNum++;
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  make_ps_file_title()
*
* MODULE FUNCTION:
*
*   This routine creates a PostScript file based on Symbol_Map contents.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  make_ps_file_title( fp )

{
    char    comp_Purpose[MAX_PURPOSE];
    int     old_scale = PrintScale;

    /*
     *  pass drawing routine the midpoint of the drawing space on the page,
     *  the comp name, and the comp purpose.
     */

    cvt_to_str_array( CompPurpose, comp_Purpose );

    PrintScale = REDUCED;

    fprintf( fp, "newpath\n" );
    fprintf( fp, "%d %d (%s) %s PrintTitlePage\n", pix_to_ps(URCX-pix_left_margin()),
       pix_to_ps((URCX+pix_left_margin())/2), CompFile, comp_Purpose );

    fprintf( fp, "showpage\n" );

    PrintScale = old_scale;
    PageNum = 1;
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  print_page()
*
* MODULE FUNCTION:
*
*   This routine adds the PostScript code to a file which prints the graphical
*   symbols.  This routine also adds the PostScript code to control the number
*   of copies.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  print_page( fp, lrcx, lrcy, xoffset, yoffset )

    FILE    *fp;
    int     lrcx,    lrcy,
            xoffset, yoffset;

{
    char                    pn[3];

    /*
     *  Clip imageable area according to parameter dimensions
     */

    fprintf( fp, "newpath\n");
    fprintf( fp, "%d %d moveto\n",  pix_to_ps(pix_left_margin()),
                                    pix_to_ps(pix_centering_height()) );
    fprintf( fp, "%d %d lineto\n",  pix_to_ps( lrcx ),
                                    pix_to_ps(pix_centering_height()) );
    fprintf( fp, "%d %d lineto\n",  pix_to_ps( lrcx),
                                    pix_to_ps(pix_centering_height()-lrcy));
    fprintf( fp, "%d %d lineto\n",  pix_to_ps(pix_left_margin()),
                                    pix_to_ps(pix_centering_height()-lrcy ));
    fprintf( fp, "%d %d lineto\n",  pix_to_ps(pix_left_margin()),
                                    pix_to_ps(pix_centering_height()) );

    fprintf( fp, "clip\nnewpath\n" );

    /*
     *  Set the number of copies.
     */

    fprintf( fp, "/#copies %d def\n", Copies );

    /*
     *  Print the symbols and their lines.
     */

    print_symbols( fp, xoffset, yoffset );

    /*
     *  convert the current page number to a string, print it.
     *  then draw the current page.
     */

    itoa( PageNum, pn );

    fprintf( fp, "(%s) EndPage\n", pn );
    PageNum++;
}
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    print.h
*
* FILE FUNCTION:
*
*   This file contains the variables and function prototype for print.c   Print.c is
*   still being developed as routines are moved into it from cbr_menu.c.  This file
*   may be very small now, but it will grow as more of the print routines are moved
*   from cbr_menu.c to print.c
*
*
* FILE MODULES:
*
*   N/A
*
*****************************************<----->*****************************************/

/*
 *  Function prototypes.
 */

void    print_page();

/*
 *  Global variables
 */

int PageNum,                /* for printing hard copies */
    Report;                 /*  are we printing a report or just a page? */
```

```
%!

/textheight 12 def

/ellipsedict 12 dict def
/circledict 12 dict def
circledict /mtrx matrix put
/boxdict 12 dict def
boxdict /mtrx matrix put
/pausedict 12 dict def
pausedict /mtrx matrix put
/stopdict 12 dict def
stopdict /mtrx matrix put
/sstartdict 12 dict def
sstartdict /mtrx matrix put
/printdict 15 dict def
printdict /mtrx matrix put
/textdict 12 dict def
textdict /mtrx matrix put
/ifdict 12 dict def
ifdict /mtrx matrix put
/elempurpdict 12 dict def
elempurpdict /mtrx matrix put
/reportboxdict 12 dict def
reportboxdict /mtrx matrix put
/setdict 12 dict def
setdict /mtrx matrix put
/epdict 12 dict def
epdict /mtrx matrix put
/titledict 12 dict def
titledict /mtrx matrix put


/EndPage                    %stack: pagenum
{
    epdict begin
    /pn exch    def

    initclip
    pn (0) ne
    {   /textheight 12 def
        /Courier findfont  12 scalefont setfont
        510 30 moveto
        (Page ) show
        pn show
        stroke } if
    showpage
    end

} def

/PrintTitlePage             %stack: width midpt name purpose
{
    titledict begin

    /purpose exch def
    /compname exch def
    /midpt exch def
    /width exch def
    /savematrix mtrx currentmatrix def

    80 740 700 480 drawBox
    midpt 500 translate

    3 SetScale

    % subtract 1/2 width of comp name from midpt,
    % move there and print compname.

    compname stringwidth pop 2 div
    neg 0 moveto
    compname show

    0 SetScale
    width 4 div neg 100 neg  width  purpose  BreakIntoLines

    savematrix setmatrix
    end


} def

/PrintElemPurpose           %stack: x y w str
{
     elempurpdict begin

    /tempstring exch def
    /wtemp exch def
    /ytemp exch def
    /xtemp exch def
    /savematrix mtrx currentmatrix def
    xtemp ytemp translate

    %print Purpose: label at x y in big font
    /textheight 12 def
    /Courier-Bold findfont  12 scalefont setfont
    0 0 moveto
    (Purpose:) show

    /textheight 10 def
    /Courier findfont  10 scalefont setfont
    2   textheight 2 mul neg    wtemp   tempstring  BreakIntoLines
    savematrix setmatrix

    end

} def


/PrintReportBox          %stack: ulcx ulcy height width str boxtype
{
    reportboxdict begin

    /boxtype exch def
    /tempstring exch def
    /wtemp exch def
    /htemp exch def
    /ytemp exch def
    /xtemp exch def

    xtemp ytemp htemp wtemp drawBox

    /savematrix mtrx currentmatrix def
    xtemp ytemp translate

    2 textheight neg wtemp tempstring BreakIntoLines

    boxtype 0 eq
```

```
    {   90 neg 10 neg moveto
        (Logical) show
        90 neg 30 neg moveto
        (Description) show } if
    boxtype 1 eq
    {   90 neg 10 neg moveto
        (Comp) show
        90 neg 30 neg moveto
        (Expression) show } if
    boxtype 2 eq
    {   90 neg 10 neg moveto
        (Comment) show } if
    savematrix setmatrix

    end
} def


/SetScale               %stack: scale
{
    /tempscale exch def

    tempscale 0 eq
    { /textheight 12 def
       /Courier findfont  12 scalefont setfont} if
    tempscale 1 eq
    { /textheight 8 def
       /Courier findfont  8 scalefont setfont} if
    tempscale 2 eq
    { /textheight 4 def
       /Courier findfont  4 scalefont setfont} if
    tempscale 3 eq
    { /textheight 20 def
       /Courier findfont  20 scalefont setfont} if
} def


/BreakIntoLines             %stack: x y width str
{
    /tempstring exch def
    /wtemp exch def
    /ytemp exch def
    /xtemp exch def
    xtemp ytemp moveto
    tempstring  {
            dup (@@) eq
            {/ytemp ytemp textheight sub def
                xtemp ytemp moveto}
            {dup stringwidth pop
            currentpoint pop
            add wtemp gt
            {/ytemp ytemp textheight sub def
                xtemp ytemp moveto} if
            show} ifelse
            } forall
    stroke
}def


/ellipse                %stack: x y radius
    { ellipsedict begin
        /radius exch def
        /y exch def
        /x exch def

        1 .5 scale
        x y 2 mul radius 0 360 arc
        stroke
        1 2 scale
        (BEGIN) stringwidth pop 2 div
        x exch sub
        y textheight 4 sub 2 div sub moveto
        (BEGIN) show
        stroke
        end
    } def

/circle                 %stack: x y radius
    { circledict begin
        /radius exch def
        /y exch def
        /x exch def

        /savematrix mtrx currentmatrix def
        x y radius 0 360 arc
        stroke
        (END) stringwidth pop 2 div
        x exch sub
        y textheight 4 sub 2 div sub moveto
        (END) show
        stroke
        savematrix setmatrix
        end
    } def


/set                    %stack: x y height width string
{ setdict begin
    /str exch def
    /w exch def
    /h exch def
    /y exch def
    /x exch def

    %/savematrix mtrx currentmatrix def
    gsave
    x y h w  drawBox
    x y translate

    2       textheight neg      w       str BreakIntoLines

    %savematrix setmatrix
    grestore

    end
} def


/drawBox                %stack: x y height width
{
    boxdict begin
    /w exch def
    /h exch def
    /y exch def
    /x exch def

    /savematrix mtrx currentmatrix def
```

```
    x y translate
    0 0 moveto
    w 0 lineto
    w h neg lineto
    0 h neg lineto
    0 0 lineto
    stroke
    savematrix setmatrix

    end
} def


/pause                  %stack: x y radius pausetime
    { pausedict begin
    /pausetime exch def
    /radius exch def
    /y exch def
    /x exch def

    /savematrix mtrx currentmatrix def
        x y radius 0 360 arc
        stroke

    (PAUSE) stringwidth pop 2 div
    x exch sub
    y textheight 4 sub add moveto
    (PAUSE) show

    pausetime stringwidth pop 2 div
    x exch sub
    y textheight sub moveto
    pausetime show

    stroke

    /textheight 6 def
    /Courier findfont 6 scalefont setfont

    x 4 sub y radius add textheight sub moveto
    (12) show
    x radius add 5 sub y moveto
    (3) show
    x radius sub y 3 sub moveto
    (9) show
    x 4 sub y radius sub 2 add moveto
    (6) show

    .5 setlinewidth
    x y radius add 7 sub moveto
    x y radius add 14 sub lineto
    x 3 sub y radius sub 10 add moveto
    x 10 sub y radius sub 6 add lineto
    stroke
    1 setlinewidth
    savematrix setmatrix

    end
    } def

/goto                   %stack: x y height width compname
    { boxdict begin
    /compname exch def
    /w exch def
    /h exch def
    /y exch def
    /x exch def

    /savematrix mtrx currentmatrix def
        x y translate

        0   h 2 div neg             moveto
        0   0       h 2 div     0   h 2 div     arcto
        0   h 2 div neg             moveto
        0   h neg   h 2 div     h neg   h 2 div     arcto
    stroke

        %top line
        h 2 div  0 moveto
        w h 2 div sub  0 lineto

        %bottom line

        h 2 div  h neg moveto
        w h 2 div sub  h neg lineto
    stroke

        w   h 2 div neg             moveto
        w   0       w h 2 div sub   0   h 2 div     arcto
        w   h 2 div neg             moveto
        w   h neg   w h 2 div sub   h neg   h 2 div     arcto

    stroke

        (CALL) stringwidth pop 2 div
        w 2 div  exch sub
        h 5 div 2 mul neg moveto
        (GOTO) show

        compname stringwidth pop 2 div
        w 2 div  exch sub
        h 5 div 3 mul neg moveto
        compname show

    stroke
    savematrix setmatrix
    end
    } def

/sstart                 %stack: x y height width compname
    { sstartdict begin
    /compname exch def
    /w exch def
    /h exch def
    /y exch def
    /x exch def

    /savematrix mtrx currentmatrix def
    x y h w  drawBox
    x y translate

        w 7 div 0 moveto
        w 7 div h neg lineto

        w w 7 div sub 0 moveto
        w w 7 div sub h neg lineto

        (ACTIVATE) stringwidth pop 2 div
```

```
        w 2 div  exch sub
        h 5 div 2 mul neg moveto
        (ACTIVATE) show

        compname stringwidth pop 2 div
        w 2 div  exch sub
        h 5 div 3 mul neg moveto
        compname show

    stroke
    savematrix setmatrix
    end
    } def

/sstop                      %stack: x y side leg compname
    { stopdict begin
    /compname exch def
    /leg exch def
    /side exch def
    /y exch def
    /x exch def
    /savematrix mtrx currentmatrix def

    x y translate
    0                   leg neg     moveto
    leg                 0           lineto
    leg side add        0           lineto
    side leg 2 mul add  leg neg     lineto

    side leg 2 mul add  leg side add neg    lineto

    leg side add        side leg 2 mul add neg  lineto
    leg                 side leg 2 mul add neg  lineto
    0                   leg side add neg        lineto

    0                   leg neg         lineto

    (STOP) stringwidth pop 2 div
    side leg 2 mul add 2 div    exch sub
    leg textheight add neg moveto
    (STOP) show

    compname stringwidth pop 2 div
    side leg 2 mul add 2 div    exch sub
    leg side add 4 sub neg moveto
    compname show

    stroke
    savematrix setmatrix
    end
    } def

/pprint                     %stack: x y height width corner1 corner2 str
    { printdict begin
    /str exch def
    /corner2 exch def
    /corner1 exch def
    /w exch def
    /h exch def
    /y exch def
    /x exch def

    /savematrix mtrx currentmatrix def
    x y translate
        0 corner1 neg moveto
        corner2 0 lineto
        w 0 lineto
        w h neg lineto
        0 h neg lineto
        0 corner1 neg lineto

    stroke
    2       corner1 textheight add neg  w   str     BreakIntoLines
    savematrix setmatrix
    end
    } def

/text                       %stack: x y textstr
{ textdict begin
    /textstr exch def
    /y exch def
    /x exch def
    %/tempstr 1 string def

    /savematrix mtrx currentmatrix def
    x y translate
    2 textheight neg 1000 textstr BreakIntoLines
        %tempstr cvs
        %(\n) eq
        %{/y y textheight sub def
        %       2 y moveto} if
        %show } forall
    savematrix setmatrix
    end
} def

/iif                        %stack: x y width height corner string
    { ifdict begin
    /str exch def
    /tc exch def
    /h exch def
    /w exch def
    /y exch def
    /x exch def

    /savematrix mtrx currentmatrix def
    x y translate

    % draw top triangle

    0           tc neg      moveto
    w 2 div     0           lineto
    w           tc neg      lineto

    % draw right side

    w           h tc sub neg    lineto

    % draw bottom triangle

    w 2 div     h neg           lineto
    0           h tc sub neg    lineto

    % draw left side

    0           tc neg      lineto

    % print text string
```

```
2       tc textheight add 2 sub neg     w   str     BreakIntoLines

savematrix setmatrix
end
} def
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    setup_palette.c
*
*
* FILE FUNCTION:
*
*   This file contains the routines which initialize the pixmaps for the palette
*   area menu.  The palette area menu contains the graphical symbols which are used
*   to build the graphical elements.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /home/project/3531/Docu/GCB.spec.doc
*
*
* FILE MODULES:
*
*   CreateDefaultImage()- This module creates an XImage with the specified fields.
*   get_my_foreground() - retrieves the foreground color of each item.
*   get_my_background() - retrieves the background color of each item.
*   set_my_background() - sets the background color of each item.
*   set_my_foreground() - sets the foreground color of each item.
*   setup_palette()     - sets the pixmaps and colors of the palette items.
*
*****************************************<----->*****************************************/

#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>

#include <Xm/Xm.h>
#include <Xm/PushB.h>
#include <Xm/Form.h>
#include <Xm/RowColumn.h>

#include "gcb.h"
#include "pixmaps.h"
#include "widgets.h"
#include "menu.h"


char    *palette_items[10] = {
        "BEGIN",
        "END",
        "IF",
        "SET",
        "PAUSE",
        "GOTO",
        "ACTIVATE",
        "STOP",
        "PRINT",
        "TEXT"
        };

/*
 *  structure for storing each palette item's fore and background colors.
 */

typedef struct
    {
    unsigned long fore, back;
    } fore_back_struct;

fore_back_struct    fore_back[NUM_PALETTE];
```

```
/*************************************<----->*************************************
*
* MODULE NAME:  CreateDefaultImage()
*
*
* MODULE FUNCTION:
*
*   This module creates an XImage with the specified fields.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*************************************<----->*************************************/

XImage  *CreateDefaultImage( bits, width, height )

    char *bits;
    int   width, height;

{
    XImage *image;

    image = (XImage *) XtMalloc( sizeof (XImage) );
    image->width        = width;
    image->height       = height;
    image->data         = bits;
    image->depth        = 1;
    image->xoffset      = 0;
    image->format       = XYBitmap;
    image->byte_order   = LSBFirst;
    image->bitmap_unit  = 8;
    image->bitmap_pad   = 8;
    image->bytes_per_line = (width+7)/8;
    image->bitmap_bit_order = LSBFirst;
    return( image );
}
```

```
/*************************************<----->*************************************
*
* MODULE NAME:  get_my_background()
*               set_my_background()
*               get_my_foreground()
*               set_my_foreground()
*
* MODULE FUNCTION:
*
*   These routines set and retrieve the fore and background colors of each item.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*************************************<----->*************************************/

unsigned long  get_my_background( item )

    int     item;

{
    return( fore_back[item].back );
}


void  set_my_background( item, back )

    int             item;
    unsigned long   back;

{
    fore_back[item].back = back;
}


unsigned long  get_my_foreground( item )

    int     item;

{
    return( fore_back[item].fore );
}


void  set_my_foreground( item, fore )

    int             item;
    unsigned long   fore;

{
    fore_back[item].fore = fore;
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  setup_palette()
*
*
* MODULE FUNCTION: This module sets the pixmaps and colors of the palette items.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  setup_palette( parent, under )

    Widget  parent, under;

    /*
     *  place the palette widget underneath the 'under' parameter.
     */

{
    XImage  *image;
    Pixmap  pixmap;
    int     i, n;
    Arg     args[10];

    /*
     *  create images for each of the palette item pixmaps;
     *  these will be retrieved when we change colors.
     */

    image = CreateDefaultImage( begin_bits, 64, 64 );
    XmInstallImage( image, "BEGIN" );
    image = CreateDefaultImage( end_bits, 64, 64 );
    XmInstallImage( image, "END" );
    image = CreateDefaultImage( if_bits, 64, 64 ) ;
    XmInstallImage( image, "IF" );
    image = CreateDefaultImage( set_bits, 64, 64 );
    XmInstallImage( image, "SET" );
    image = CreateDefaultImage( pause_bits, 64, 64 );
    XmInstallImage( image, "PAUSE" );
    image = CreateDefaultImage( call_bits, 64, 64 );
    XmInstallImage( image, "GOTO" );
    image = CreateDefaultImage( activate_bits, 64, 64 );
    XmInstallImage( image, "ACTIVATE" );
    image = CreateDefaultImage( stop_bits, 64, 64 );
    XmInstallImage( image, "STOP" );
    image = CreateDefaultImage( print_bits, 64, 64 );
    XmInstallImage( image, "PRINT" );
    image = CreateDefaultImage( text_bits, 64, 64 );
    XmInstallImage( image, "TEXT" );

    /*
     *  create Frame to hold palette rowcol widget
     */

    n = 0;

    /*
     *  palette frame is not mapped when managed, only when we are not
     *  creating an expression.
     */

    XtSetArg( args[n], XtNmappedWhenManaged, FALSE ); n++;
    XtSetArg( args[n], XmNbottomAttachment,  XmATTACH_FORM ); n++;
    frame_palette = (Widget) XmCreateFrame( parent, "frame_palette", args, n );
    XtManageChild( frame_palette );
    set_attach_widget( frame_palette, under, NULL, NULL, NULL );

    /*
     *  create form to hold rowcol palette
     */

    n = 0;
    XtSetArg( args[n], XmNwidth, 278 ); n++;
    XtSetArg( args[n], XmNheight, 450 );  n++;
    FormW = XmCreateForm( frame_palette, NULLS, args, n );
    XtManageChild( FormW );

    /*
     *  create rowcol to hold palette items
     */

    n = 0;
    XtSetArg( args[n], XmNpacking,          XmPACK_NONE          ); n++;
    XtSetArg( args[n], XmNnumColumns,       2                    ); n++;
    XtSetArg( args[n], XmNentryAlignment,   XmALIGNMENT_CENTER   ); n++;
    XtSetArg( args[n], XmNorientation,      XmVERTICAL           ); n++;
    rc_palette = XmCreateRowColumn( FormW, "rc_palette", args, n );
    XtManageChild( rc_palette );

    /*
     *  find pixmap for each button, set callback function, and
     *  create button.
     */

    for ( i=0; i < 10; i++ )
        {
        pixmap = XmGetPixmap( XtScreen(rc_palette), palette_items[i],
                              colors[MAX_COLORS-2], 0 );

        /*
         *  create push button with proper pixmap and width and height.
         */

        n = 0;
        XtSetArg( args[n], XmNlabelType, XmPIXMAP ); n++;
        XtSetArg( args[n], XmNlabelPixmap, pixmap ); n++;
        XtSetArg( args[n], XmNwidth,  64 ); n++;
        XtSetArg( args[n], XmNheight, 64 ); n++;

        /*
         *  Record the palette item's type in userData
         */

        XtSetArg( args[n], XmNuserData, i ); n++;

        /*
         *  place palette items in 2 columns of 5 each
         */

        if ( i < 5 )
            {
            XtSetArg( args[n], XmNx, 40 ); n++;
```

```
        XtSetArg( args[n], XmNy, (85 * i) + 15 ); n++;
        }
    else
        {
        XtSetArg( args[n], XmNx, 165 ); n++;
        XtSetArg( args[n], XmNy, (85 * (i-5)) + 15 ); n++;
        }

    Palette[i] = XmCreatePushButton( rc_palette, NULLS, args, n );
    XtManageChild( Palette[i] );

    switch ( i )
        {

        /*
         *  set callback depending on type of button.
         */

        case 0:
        case 1:
            XtAddCallback( Palette[i], XmNactivateCallback, cbr_palette, i );
            break;
        case 2:
        case 3:
            XtAddCallback( Palette[i], XmNactivateCallback, cbr_if, i );
            break;
        default:
            XtAddCallback( Palette[i], XmNactivateCallback, cbr_printset, i );
            break;
        }

    /*
     *  Set initial default colors.
     */

    set_my_foreground( i, MAX_COLORS-2 );
    set_my_background( i, 0 );
    }
}
```

```
/*
 *  Various COTS include files.
 */

#include <stdio.h>
#include <math.h>
#include <memory.h>
#include <sys/stdtypes.h>
#include <sys/param.h>
#include <sys/signal.h>

/*
 *  Various Custom include files.
 */

#include <matrix.h>
#include <skeleton_element.h>

/*
 *  Local data structure to store process information.  The size of the table is
 *  equal to the maximum number of processes that a single user can have (which
 *  implies that the table will never fill up.
 */

static struct process_data process_table[ MAXUPRC ];
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  initialize_process_table()
*
*
* MODULE FUNCTION:
*
*  Function initializes the process table which is maintained by the running Comp.
*  The process table is used to keep track of other Comps which are spawned/ACTIVATED
*  by the current Comp.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

void initialize_process_table()
{
    register int i;

    for ( i = 0; i < MAXUPRC; i++ )
        process_table[ i ].pd_process_status = NO_PROCESS_DEFINED;
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  start_process()
*
*
* MODULE FUNCTION:
*
*  Function start_process in invoked when a COMP is to be initiated.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void start_process( process_name )

    char *process_name;

{
    register int i;
    pid_t process_id;

    /*
     *  Search the process table to determine if an entry exists
     *  for the new processed to be spawned.
     */

    for ( i = 0;
          ( ( i < MAXUPRC ) &&
            ( process_table[ i ].pd_process_status & PROCESS_STARTED ) );
          i++ );

    /*
     *  Report an error if an entry could not be found.
     */

    if ( i == MAXUPRC) {
        fprintf( stderr, "You have filled up the process table...\n" );
        return;
    }

    /*
     *  Attempt to FORK and EXEC the new process.  If successful, record
     *  the process ID in the process table.
     */

    process_id = fork();
    if ( process_id == -1 )
        fprintf( stderr, "Could not spawn a new process\n" );
    else
        if ( process_id == 0 ) {
            execl( process_name, NULL, NULL );

            /*
             *  The process was not successfully started, exit this process
             *  as no value added has occurred.
             */

            fprintf( stderr, "Process %s could not be EXEC'ed\n", process_name );
            exit( -1 );
        }
        else {

            /*
             *  Record the vital statistics of the process.  It would
             *  be nice to install a heartbeat so that it could be
             *  100% determined that the process (the EXEX'ed one) was
             *  really started.
             */

            strcpy( process_table[ i ].pd_process_name, process_name );
            process_table[ i ].pd_process_id = process_id;
            process_table[ i ].pd_process_status |= PROCESS_STARTED;
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  stop_process()
*
*
* MODULE FUNCTION:
*
*  Function stop_process is invoked when a previously started COMP is to be
*  terminated.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

void stop_process( process_name )

    char *process_name;

{
    register int i, index;

    /*
     *  Search each entry in the process status table for the
     *  specified process name.  If the entry exists, find the
     *  most recent.
     */

    for ( i = 0, index = -1; i < MAXUPRC; i++ )

        /*
         *  If the process table entry is defined determine if
         *  the process ID is the most recent (assuming that the
         *  most recent will always have the largest UNIX PID).
         */

        if ( ( process_table[ i ].pd_process_status & PROCESS_STARTED ) &&
             ( strcmp( process_name,
                       process_table[ i ].pd_process_name ) == 0 ) )
            if ( index == -1 )
                index = i;
            else
                if ( process_table[ i ].pd_process_id >
                     process_table[ index ].pd_process_id )
                    index = i;

    /*
     *  Terminate the process if it was found in the process table.
     *  Mark the entry in the process table as not active.
     */

    if ( index != -1 )
        if ( kill( process_table[ index ].pd_process_id, SIGKILL ) == 0 )
            process_table[ index ].pd_process_status = NO_PROCESS_DEFINED;
        else
            fprintf( stderr, "The process: %s could not be terminated\n",
                process_name );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  bind_ws_global()
*
*
* MODULE FUNCTION:
*
*  Function bind_ws_global is invoked to bind the address of a workstation global
*  variable to that of an address in data acquisition.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

int bind_ws_global( global_name, data_type, pointer )

    char *global_name;
    int data_type;
    void *pointer;

{
    /*
     *  Appropriate code must be added to bind these variables to a real memory
     *  location when the GCB is fullu implemented.
     */
}
```

C-5

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  bind_ws_object()
*
*
* MODULE FUNCTION:
*
*  Function bind_object is invoked to bind the address of an object global global
*  variable to that of an address in data acquisition.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

int bind_ws_object( object_name, data_type, pointer )

    char *object_name;
    int data_type;
    void *pointer;

{
    /*
     *  Appropriate code must be added to bind these variables to a real memory
     *  location when the GCB is fullu implemented.
     */
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  ci()
*
*
* MODULE FUNCTION:
*
*  Function ci is invoked to compute the index into a multi-dimensional array while
*  treating the base pointer as a single dimensioned array.  This is done because of
*  the difficulties of dealing with variable sized two dimensional arrays in C.  Note
*  that this function (and all the matrix functions) assume traditional C based
*  arrays which are addressed starting at position zero (0).
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

static int ci( i, j, number_rows )

    int i, j, number_rows;

{
    return( j + ( i * number_rows ) );
}  /*  of function  */
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  matrix_init()
*
*
* MODULE FUNCTION:
*
*  Function matrix_init is invoked to initialize a matrix to a specified value.  The
*  value to be initialized is ALWAYS as passed as double and then convert as necessary
*  to the target data type.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

void matrix_init( data_type, matrix, number_rows, number_columns, init_value )

    int data_type, number_rows, number_columns;
    void *matrix;
    double init_value;

{
    register int i, j;
    int *int_matrix, int_value;
    float *float_matrix, float_value;
    double *double_matrix, double_value;
    short *short_matrix, short_value;
    unsigned int *unsigned_matrix, unsigned_value;

    switch( data_type ) {
        case INT    : int_matrix = (int *)matrix;
                      int_value = (int)init_value;
                      for ( i = 0; i < number_rows; i++ )
                        for ( j = 0; j < number_columns; j++ )
                            int_matrix[ ci( i, j, number_rows ) ] = int_value;

                      break;
        case FLOAT  : float_matrix = (float *)matrix;
                      float_value = (float)init_value;
                      for ( i = 0; i < number_rows; i++ )
                        for ( j = 0; j < number_columns; j++ )
                            float_matrix[ ci( i, j, number_rows ) ] = float_value;

                      break;
        case DOUBLE : double_matrix = (double *)matrix;
                      double_value = (double)init_value;

                      for ( i = 0; i < number_rows; i++ )
                        for ( j = 0; j < number_columns; j++ )
                            double_matrix[ ci( i, j, number_rows ) ] = double_value;

                      break;
        case SHORT  : short_matrix = (short *)matrix;
                      short_value = (short)init_value;

                      for ( i = 0; i < number_rows; i++ )
                        for ( j = 0; j < number_columns; j++ )
                            short_matrix[ ci( i, j, number_rows ) ] = short_value;
```

```
                break;
        case UNSIGN : unsigned_matrix = (unsigned int *)matrix;
                unsigned_value = (unsigned int)init_value;

                for ( i = 0; i < number_rows; i++ )
                  for ( j = 0; j < number_columns; j++ )
                      unsigned_matrix[ ci( i, j, number_rows ) ] =
                          unsigned_value;

                break;
    }  /*  of switch  */
}  /*  of function  */
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  ci4()
*
*
* MODULE FUNCTION:
*
*  Function ci4 is invoked to compute the index into a multi-dimensional array while
*  treating the base pointer as a single dimensioned array.  This is done because of
*  the difficulties of dealing with variable sized two dimensional arrays in C.
*  Note that this function (and all the matrix functions) assume traditional C based
*  arrays which are addressed starting at position zero (0).
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

static int ci4( i, j, k, l, sub1, sub2, sub3 )

    int i, j, sub1;

{
    return( l + ( i * sub1 ) + ( j * sub2 ) + ( k * sub3 ) );
}  /*  of function  */
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  matrix_init4()
*
*
* MODULE FUNCTION:
*
*  Function matrix_init4 is invoked to initialize a matrix to a specified value.
*  The value to be initialized is ALWAYS as passed as double and then convert as
*   necessary to the target data type.  This function is for 3D and 4D arrays only.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

void matrix_init4( data_type, matrix, sub1, sub2, sub3, sub4, init_value )

    int data_type, sub1, sub2, sub3, sub4;
    void *matrix;
    double init_value;

{
    register int i, j, k, l;
    int *int_matrix, int_value;
    float *float_matrix, float_value;
    double *double_matrix, double_value;
    short *short_matrix, short_value;
    unsigned int *unsigned_matrix, unsigned_value;

    switch( data_type ) {
        case INT    : int_matrix = (int *)matrix;
                      int_value = (int)init_value;
                      for ( i = 0; i < sub1; i++ )
                        for ( j = 0; j < sub2; j++ )
                          for ( k = 0; k < sub3; k++ )
                            for ( l = 0; l < sub4; l++ )
                              int_matrix[ ci4( i, j, k, l, sub1, sub2, sub3 ) ] =
                                int_value;

                      break;
        case FLOAT  : float_matrix = (float *)matrix;
                      float_value = (float)init_value;
                      for ( i = 0; i < sub1; i++ )
                        for ( j = 0; j < sub2; j++ )
                          for ( k = 0; k < sub3; k++ )
                            for ( l = 0; l < sub4; l++ )
                              float_matrix[ ci4( i, j, k, l, sub1, sub2, sub3 ) ] =
                                    float_value;

                      break;
        case DOUBLE : double_matrix = (double *)matrix;
                      double_value = (double)init_value;

                      for ( i = 0; i < sub1; i++ )
                        for ( j = 0; j < sub2; j++ )
                          for ( k = 0; k < sub3; k++ )
                            for ( l = 0; l < sub4; l++ )
                              double_matrix[ ci4( i, j, k, l, sub1, sub2, sub3 ) ] =
                                double_value;

                      break;
        case SHORT  : short_matrix = (short *)matrix;
                      short_value = (short)init_value;

                      for ( i = 0; i < sub1; i++ )
                        for ( j = 0; j < sub2; j++ )
                          for ( k = 0; k < sub3; k++ )
                            for ( l = 0; l < sub4; l++ )
                              short_matrix[ ci4( i, j, k, l, sub1, sub2, sub3 ) ] =
                                short_value;

                      break;
        case UNSIGN : unsigned_matrix = (unsigned int *)matrix;
                      unsigned_value = (unsigned int)init_value;

                      for ( i = 0; i < sub1; i++ )
                        for ( j = 0; j < sub2; j++ )
                          for ( k = 0; k < sub3; k++ )
                            for ( l = 0; l < sub4; l++ )
                              unsigned_matrix[ ci4( i, j, k, l, sub1, sub2, sub3 ) ]
                                = unsigned_value;

                      break;
    }  /*  of switch  */
}  /*  of function  */
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  matrix_copy4()
*
*
* MODULE FUNCTION:
*
*  Function matrix_copy4 will copy one matrix to another.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

void matrix_copy4( data_type, matrix1, matrix2, sub1, sub2, sub3, sub4 )

    int data_type, sub1, sub2, sub3, sub4;
    void *matrix1, *matrix2;

{
    register int num_bytes;

    /*
     *  When this routine is invoked, we are assured of at least a 3D array, the 4th
     *  dimension may be zero so it is added separtely.
     */

    switch( data_type ) {
        case INT    : num_bytes = ( sub1 + sub2 + sub3 ) * sizeof( int ) +
                                  sub4 * sizeof( int );
                      break;
        case FLOAT  : num_bytes = ( sub1 + sub2 + sub3 ) * sizeof( float ) +
                                  sub4 * sizeof( float );
                      break;
        case DOUBLE : num_bytes = ( sub1 + sub2 + sub3 ) * sizeof( double ) +
                                  sub4 * sizeof( double );
                      break;
        case SHORT  : num_bytes = ( sub1 + sub2 + sub3 ) * sizeof( short ) +
                                  sub4 * sizeof( short );
                      break;
        case UNSIGN : num_bytes = ( sub1 + sub2 + sub3 ) * sizeof( unsigned ) +
                                  sub4 * sizeof( unsigned );
                      break;
    }  /*  of switch  */

    /*
     *  Copy the bytes over from the source pointer to the destination pointer.
     */

    memcpy( matrix2, matrix1, num_bytes );
}  /*  of function  */
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  matrix_print()
*
*
* MODULE FUNCTION:
*
*  Function matrix_print is invoked to print the contents of a matrix.  It is not
*  currently callable from a GCB statment but it is included for future enhancements
*  to the GCB.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

void matrix_print( data_type, matrix, number_rows, number_cols )

    int data_type, number_rows, number_cols;
    void *matrix;

{
    register int i, j;
    int *int_matrix;
    float *float_matrix;
    double *double_matrix;
    short *short_matrix;
    unsigned int *unsigned_matrix;

    switch( data_type ) {
        case INT    : int_matrix = (int *)matrix;

                      for ( i = 0; i < number_rows; i++ ) {
                        for ( j = 0; j < number_cols; j++ )
                            fprintf( stdout, "%8d  ",
                                int_matrix[ ci( i, j, number_rows ) ] );
                        fprintf( stdout, "\n" );
                      }  /*  of for  */
                      break;
        case FLOAT  : float_matrix = (float *)matrix;

                      for ( i = 0; i < number_rows; i++ ) {
                        for ( j = 0; j < number_cols; j++ )
                            fprintf( stdout, "%8.2f  ",
                                float_matrix[ ci( i, j, number_rows ) ] );
                        fprintf( stdout, "\n" );
                      }  /*  of for  */
                      break;
        case DOUBLE : double_matrix = (double *)matrix;

                      for ( i = 0; i < number_rows; i++ ) {
                        for ( j = 0; j < number_cols; j++ )
                            fprintf( stdout, "%8.2lf  ",
                                    double_matrix[ ci( i, j, number_rows ) ] );
                        fprintf( stdout, "\n" );
                      }  /*  of for  */
                      break;
        case SHORT  : short_matrix = (short *)matrix;
```

```
                    for ( i = 0; i < number_rows; i++ ) {
                      for ( j = 0; j < number_cols; j++ )
                          fprintf( stdout, "%8d  ",
                              short_matrix[ ci( i, j, number_rows ) ] );
                      fprintf( stdout, "\n" );
                    }  /*  of for  */
                    break;
        case UNSIGN : unsigned_matrix = (unsigned int *)matrix;

                    for ( i = 0; i < number_rows; i++ ) {
                      for ( j = 0; j < number_cols; j++ )
                          fprintf( stdout, "%8d  ",
                              unsigned_matrix[ ci( i, j, number_rows ) ] );
                      fprintf( stdout, "\n" );
                    }  /*  of for  */
                    break;
    }  /*  of switch  */
}  /*  of function  */
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  matrix_dot()
*
*
* MODULE FUNCTION:
*
*  Function matrix_dot will perform a dot product of two matrices.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

void matrix_dot( data_type, matrix1, matrix2, number_rows, dotprod )

    int data_type, number_rows;
    void *matrix1, *matrix2, *dotprod;

{
    int *int_m1, *int_m2, int_dotprod, i;
    float *float_m1, *float_m2, float_dotprod;
    double *double_m1, *double_m2, double_dotprod;
    short *short_m1, *short_m2, short_dotprod;
    unsigned int *unsigned_m1, *unsigned_m2, unsigned_dotprod;

    switch( data_type ) {
        case INT    : int_dotprod = 0;
                      int_m1 = (int *)matrix1;
                      int_m2 = (int *)matrix2;

                      for ( i = 0; i < number_rows; i++ )
                        int_dotprod += int_m1[ ci( i, 0, number_rows ) ] *
                                       int_m2[ ci( i, 0, number_rows ) ];

                      *(int *)dotprod = int_dotprod;
                      break;
        case FLOAT  : float_dotprod = 0.0;
                      float_m1 = (float *)matrix1;
                      float_m2 = (float *)matrix2;

                      for ( i = 0; i < number_rows; i++ )
                        float_dotprod += float_m1[ ci( i, 0, number_rows ) ] *
                                         float_m2[ ci( i, 0, number_rows ) ];

                      *(float *)dotprod = float_dotprod;
                      break;
        case DOUBLE : double_dotprod = 0.0;
                      double_m1 = (double *)matrix1;
                      double_m2 = (double *)matrix2;

                      for ( i = 0; i < number_rows; i++ )
                        double_dotprod += double_m1[ ci( i, 0, number_rows ) ] *
                                          double_m2[ ci( i, 0, number_rows ) ];

                      *(double *)dotprod = double_dotprod;
                      break;
        case SHORT  : short_dotprod = 0.0;
                      short_m1 = (short *)matrix1;
```

```
                short_m2 = (short *)matrix2;

                for ( i = 0; i < number_rows; i++ )
                  short_dotprod += short_m1[ ci( i, 0, number_rows ) ] *
                                   short_m2[ ci( i, 0, number_rows ) ];

                *(short *)dotprod = short_dotprod;
                break;
      case UNSIGN : unsigned_dotprod = 0.0;
                unsigned_m1 = (unsigned int *)matrix1;
                unsigned_m2 = (unsigned int *)matrix2;

                for ( i = 0; i < number_rows; i++ )
                  unsigned_dotprod += unsigned_m1[ ci( i, 0, number_rows ) ] *
                                   unsigned_m2[ ci( i, 0, number_rows ) ];

                *(unsigned int *)dotprod = unsigned_dotprod;
                break;
    } /*  of switch  */
} /*  of function  */
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  matrix_copy()
*
*
* MODULE FUNCTION:
*
*  Function matrix_copy will copy one matrix to another.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

void matrix_copy( data_type, matrix1, matrix2, sub1, sub2 )

    int data_type, sub1, sub2;
    void *matrix1, *matrix2;

{
    register int num_bytes;

    switch( data_type ) {
        case INT    : num_bytes = sub1 * sizeof( int ) + sub2 * sizeof( int );
                      break;
        case FLOAT  : num_bytes = sub1 * sizeof( float ) + sub2 * sizeof( float );
                      break;
        case DOUBLE : num_bytes = sub1 * sizeof( double ) + sub2 * sizeof( double );
                      break;
        case SHORT  : num_bytes = sub1 * sizeof( short ) + sub2 * sizeof( short );
                      break;
        case UNSIGN : num_bytes = sub1 * sizeof( unsigned )+sub2*sizeof( unsigned );
                      break;
    } /*  of switch  */

    /*
     *  Copy the bytes over from the source pointer to the destination pointer.
     */

    memcpy( matrix2, matrix1, num_bytes );
} /*  of function  */
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  matrix_cross()
*
*
* MODULE FUNCTION:
*
*  Function matrix_cross will perform the cross product of two matrices and stored
*  the result in a third matrix.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

void matrix_cross( data_type, num_rows, matrix1, matrix2, matrix3 )

    int data_type, num_rows;
    void *matrix1, *matrix2, *matrix3;

{
    int *int_m1, *int_m2, *int_result, row;
    float *float_m1, *float_m2, *float_result;
    double *double_m1, *double_m2, *double_result;
    short *short_m1, *short_m2, *short_result;
    unsigned int *unsigned_m1, *unsigned_m2, *unsigned_result;

    switch( data_type ) {
        case INT    : int_m1 = (int *)matrix1;
                      int_m2 = (int *)matrix2;
                      int_result = (int *)matrix3;

                      int_result[ ci( 0, 0, num_rows ) ] =
                        int_m1[ ci( 1, 0, num_rows ) ] *
                        int_m2[ ci( 2, 0, num_rows ) ] -
                        int_m1[ ci( 2, 0, num_rows ) ] *
                        int_m2[ ci( 1, 0, num_rows ) ];

                      int_result[ ci( 1, 0, num_rows ) ] =
                        int_m1[ ci( 2, 0, num_rows ) ] *
                        int_m2[ ci( 0, 0, num_rows ) ] -
                        int_m1[ ci( 0, 0, num_rows ) ] *
                        int_m2[ ci( 2, 0, num_rows ) ];

                      int_result[ ci( 2, 0, num_rows ) ] =
                        int_m1[ ci( 0, 0, num_rows ) ] *
                        int_m2[ ci( 1, 0, num_rows ) ] -
                        int_m1[ ci( 1, 0, num_rows ) ] *
                        int_m2[ ci( 0, 0, num_rows ) ];
                      break;

        case FLOAT  : float_m1 = (float *)matrix1;
                      float_m2 = (float *)matrix2;
                      float_result = (float *)matrix3;

                      float_result[ ci( 0, 0, num_rows ) ] =
                        float_m1[ ci( 1, 0, num_rows ) ] *
                        float_m2[ ci( 2, 0, num_rows ) ] -
                        float_m1[ ci( 2, 0, num_rows ) ] *
                        float_m2[ ci( 1, 0, num_rows ) ];

                      float_result[ ci( 1, 0, num_rows ) ] =
                        float_m1[ ci( 2, 0, num_rows ) ] *
                        float_m2[ ci( 0, 0, num_rows ) ] -
                        float_m1[ ci( 0, 0, num_rows ) ] *
                        float_m2[ ci( 2, 0, num_rows ) ];

                      float_result[ ci( 2, 0, num_rows ) ] =
                        float_m1[ ci( 0, 0, num_rows ) ] *
                        float_m2[ ci( 1, 0, num_rows ) ] -
                        float_m1[ ci( 1, 0, num_rows ) ] *
                        float_m2[ ci( 0, 0, num_rows ) ];
                      break;

        case DOUBLE : double_m1 = (double *)matrix1;
                      double_m2 = (double *)matrix2;
                      double_result = (double *)matrix3;

                      double_result[ ci( 0, 0, num_rows ) ] =
                        double_m1[ ci( 1, 0, num_rows ) ] *
                        double_m2[ ci( 2, 0, num_rows ) ] -
                        double_m1[ ci( 2, 0, num_rows ) ] *
                        double_m2[ ci( 1, 0, num_rows ) ];

                      double_result[ ci( 1, 0, num_rows ) ] =
                        double_m1[ ci( 2, 0, num_rows ) ] *
                        double_m2[ ci( 0, 0, num_rows ) ] -
                        double_m1[ ci( 0, 0, num_rows ) ] *
                        double_m2[ ci( 2, 0, num_rows ) ];

                      double_result[ ci( 2, 0, num_rows ) ] =
                        double_m1[ ci( 0, 0, num_rows ) ] *
                        double_m2[ ci( 1, 0, num_rows ) ] -
                        double_m1[ ci( 1, 0, num_rows ) ] *
                        double_m2[ ci( 0, 0, num_rows ) ];
                      break;
        case SHORT  : short_m1 = (short *)matrix1;
                      short_m2 = (short *)matrix2;
                      short_result = (short *)matrix3;

                      short_result[ ci( 0, 0, num_rows ) ] =
                        short_m1[ ci( 1, 0, num_rows ) ] *
                        short_m2[ ci( 2, 0, num_rows ) ] -
                        short_m1[ ci( 2, 0, num_rows ) ] *
                        short_m2[ ci( 1, 0, num_rows ) ];

                      short_result[ ci( 1, 0, num_rows ) ] =
                        short_m1[ ci( 2, 0, num_rows ) ] *
                        short_m2[ ci( 0, 0, num_rows ) ] -
                        short_m1[ ci( 0, 0, num_rows ) ] *
                        short_m2[ ci( 2, 0, num_rows ) ];

                      short_result[ ci( 2, 0, num_rows ) ] =
                        short_m1[ ci( 0, 0, num_rows ) ] *
                        short_m2[ ci( 1, 0, num_rows ) ] -
                        short_m1[ ci( 1, 0, num_rows ) ] *
                        short_m2[ ci( 0, 0, num_rows ) ];
                      break;
        case UNSIGN : unsigned_m1 = (unsigned int *)matrix1;
                      unsigned_m2 = (unsigned int *)matrix2;
                      unsigned_result = (unsigned int *)matrix3;

                      unsigned_result[ ci( 0, 0, num_rows ) ] =
```

```
                    unsigned_m1[ ci( 1, 0, num_rows ) ] *
                    unsigned_m2[ ci( 2, 0, num_rows ) ] -
                    unsigned_m1[ ci( 2, 0, num_rows ) ] *
                    unsigned_m2[ ci( 1, 0, num_rows ) ];

                  unsigned_result[ ci( 1, 0, num_rows ) ] =
                    unsigned_m1[ ci( 2, 0, num_rows ) ] *
                    unsigned_m2[ ci( 0, 0, num_rows ) ] -
                    unsigned_m1[ ci( 0, 0, num_rows ) ] *
                    unsigned_m2[ ci( 2, 0, num_rows ) ];

                  unsigned_result[ ci( 2, 0, num_rows ) ] =
                    unsigned_m1[ ci( 0, 0, num_rows ) ] *
                    unsigned_m2[ ci( 1, 0, num_rows ) ] -
                    unsigned_m1[ ci( 1, 0, num_rows ) ] *
                    unsigned_m2[ ci( 0, 0, num_rows ) ];
                  break;
    } /* of switch */
} /* of function */
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  matrix_ident()
*
*
* MODULE FUNCTION:
*
*  Function matrix_ident will initialize the given matrix to an identity matrix.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

void matrix_ident( data_type, size, matrix )

    int data_type, size;
    void *matrix;

{
    register int i, j;
    int *int_matrix;
    float *float_matrix;
    double *double_matrix;
    short *short_matrix;
    unsigned int *unsigned_matrix;

    switch( data_type ) {
        case INT    : int_matrix = (int *)matrix;

                      for ( i = 0; i < size; i++ )
                        for ( j = 0; j < size; j++ )
                            if ( i == j )
                                int_matrix[ ci( i, j, size ) ] = 1;
                            else
                                int_matrix[ ci( i, j, size ) ] = 0;
                      break;
        case FLOAT  : float_matrix = (float *)matrix;

                      for ( i = 0; i < size; i++ )
                        for ( j = 0; j < size; j++ )
                            if ( i == j )
                                float_matrix[ ci( i, j, size ) ] = 1.0;
                            else
                                float_matrix[ ci( i, j, size ) ] = 0.0;
                      break;
        case DOUBLE : double_matrix = (double *)matrix;

                      for ( i = 0; i < size; i++ )
                        for ( j = 0; j < size; j++ )
                            if ( i == j )
                                double_matrix[ ci( i, j, size ) ] = 1.0;
                            else
                                double_matrix[ ci( i, j, size ) ] = 0.0;
                      break;
        case SHORT  : short_matrix = (short *)matrix;

                      for ( i = 0; i < size; i++ )
                        for ( j = 0; j < size; j++ )
```

```
                    if ( i == j )
                        short_matrix[ ci( i, j, size ) ] = 1.0;
                    else
                        short_matrix[ ci( i, j, size ) ] = 0.0;
                break;
        case UNSIGN : unsigned_matrix = (unsigned int *)matrix;

                for ( i = 0; i < size; i++ )
                  for ( j = 0; j < size; j++ )
                    if ( i == j )
                        unsigned_matrix[ ci( i, j, size ) ] = 1.0;
                    else
                        unsigned_matrix[ ci( i, j, size ) ] = 0.0;
                break;
    } /* of switch */
} /* of function */
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  sub_matrix()
*
*
* MODULE FUNCTION:
*
*  Function sub_matrix will either add or subtract two matrices leaving the result
*  in a third matrix.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

static void sub_matrix( data_type, matrix1, matrix2, number_rows, number_cols,
    matrix3, what )

    int data_type, number_rows, number_cols, what;
    void *matrix1, *matrix2, *matrix3;

{
    register int i, j;
    int *int_m1, *int_m2, *int_result, row;
    float *float_m1, *float_m2, *float_result;
    double *double_m1, *double_m2, *double_result;
    short *short_m1, *short_m2, *short_result;
    unsigned int *unsigned_m1, *unsigned_m2, *unsigned_result;

    switch( data_type ) {
        case INT    : int_m1 = (int *)matrix1;
                      int_m2 = (int *)matrix2;
                      int_result = (int *)matrix3;

                      for ( i = 0; i < number_rows; i++ )
                        for ( j = 0; j < number_cols; j++ )
                            switch( what ) {
                                case ADD      :
                                        int_result[ ci( i, j, number_rows ) ] =
                                            int_m1[ ci( i, j, number_rows ) ] +
                                            int_m2[ ci( i, j, number_rows ) ];
                                       break;
                                case SUBTRACT :
                                        int_result[ ci( i, j, number_rows ) ] =
                                            int_m1[ ci( i, j, number_rows ) ] -
                                            int_m2[ ci( i, j, number_rows ) ];
                                       break;
                            } /* of switch */
                      break;
        case FLOAT  : float_m1 = (float *)matrix1;
                      float_m2 = (float *)matrix2;
                      float_result = (float *)matrix3;

                      for ( i = 0; i < number_rows; i++ )
                        for ( j = 0; j < number_cols; j++ )
                            switch( what ) {
                                case ADD      :
                                        float_result[ ci( i, j, number_rows ) ] =
                                            float_m1[ ci( i, j, number_rows ) ] +
```

```
                                    float_m2[ ci( i, j, number_rows ) ];
                                break;
                        case SUBTRACT :
                                float_result[ ci( i, j, number_rows ) ] =
                                    float_m1[ ci( i, j, number_rows ) ] -
                                    float_m2[ ci( i, j, number_rows ) ];
                                break;
                    }  /*  of switch  */
                break;
    case DOUBLE : double_m1 = (double *)matrix1;
                double_m2 = (double *)matrix2;
                double_result = (double *)matrix3;

                for ( i = 0; i < number_rows; i++ )
                  for ( j = 0; j < number_cols; j++ )
                      switch( what ) {
                          case ADD      :
                                  double_result[ ci( i, j, number_rows ) ] =
                                          double_m1[ ci( i, j, number_rows ) ] +
                                          double_m2[ ci( i, j, number_rows ) ];
                                  break;
                          case SUBTRACT :
                                  double_result[ ci( i, j, number_rows ) ] =
                                      double_m1[ ci( i, j, number_rows ) ] -
                                      double_m2[ ci( i, j, number_rows ) ];
                                  break;
                      }  /*  of switch  */
                break;
    case SHORT  : short_m1 = (short *)matrix1;
                short_m2 = (short *)matrix2;
                short_result = (short *)matrix3;

                for ( i = 0; i < number_rows; i++ )
                  for ( j = 0; j < number_cols; j++ )
                      switch( what ) {
                          case ADD      :
                                  short_result[ ci( i, j, number_rows ) ] =
                                          short_m1[ ci( i, j, number_rows ) ] +
                                          short_m2[ ci( i, j, number_rows ) ];
                                  break;
                          case SUBTRACT :
                                  short_result[ ci( i, j, number_rows ) ] =
                                      short_m1[ ci( i, j, number_rows ) ] -
                                      short_m2[ ci( i, j, number_rows ) ];
                                  break;
                      }  /*  of switch  */
                break;
    case UNSIGN : unsigned_m1 = (unsigned int *)matrix1;
                unsigned_m2 = (unsigned int *)matrix2;
                unsigned_result = (unsigned int *)matrix3;

                for ( i = 0; i < number_rows; i++ )
                  for ( j = 0; j < number_cols; j++ )
                      switch( what ) {
                          case ADD      :
                                  unsigned_result[ ci( i, j, number_rows ) ] =
                                      unsigned_m1[ ci( i, j, number_rows ) ] +
                                      unsigned_m2[ ci( i, j, number_rows ) ];
                                  break;
                          case SUBTRACT :
                                  unsigned_result[ ci( i, j, number_rows ) ] =
                                      unsigned_m1[ ci( i, j, number_rows ) ] -
                                      unsigned_m2[ ci( i, j, number_rows ) ];
                                  break;
                                  ...
                      }  /*  of switch  */
                break;
    }  /*  of switch  */
}  /*  of function  */
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  matrix_add()
*
*
* MODULE FUNCTION:
*
*  Function matrix_add will add two matrices and leave the result in a third matrix.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

void matrix_add( data_type, matrix1, matrix2, number_rows, number_cols, matrix3 )

    int data_type, number_rows, number_cols;
    void *matrix1, *matrix2, *matrix3;

{
    sub_matrix( data_type, matrix1, matrix2, number_rows, number_cols,matrix3, ADD );
}  /*  of function  */
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  matrix_sub()
*
*
* MODULE FUNCTION:
*
*  Function matrix_sub will subtract two matrices and leave the result in a
*  third matrix.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

void matrix_sub( data_type, matrix1, matrix2, number_rows, number_cols, matrix3 )

    int data_type, number_rows, number_cols;
    void *matrix1, *matrix2, *matrix3;

{
    sub_matrix( data_type, matrix1, matrix2, number_rows, number_cols,matrix3,
        SUBTRACT );
}  /*  of function  */
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  matrix_mult()
*
*
* MODULE FUNCTION:
*
*  Function matrix_mult will multiple two matrices and store the result in a
*  third matrix.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

void matrix_mult( data_type, matrix1, matrix2, number_rows, number_cols1,
                  number_cols2, matrix3 )

    int data_type, number_rows, number_cols1, number_cols2;
    void *matrix1, *matrix2, *matrix3;

{
    register int i, j, k;
    int *int_m1, *int_m2, *int_result, row;
    float *float_m1, *float_m2, *float_result;
    double *double_m1, *double_m2, *double_result;
    short *short_m1, *short_m2, *short_result;
    unsigned int *unsigned_m1, *unsigned_m2, *unsigned_result;

    switch( data_type ) {
        case INT    : int_m1 = (int *)matrix1;
                      int_m2 = (int *)matrix2;
                      int_result = (int *)matrix3;
                      for ( i = 0; i < number_rows; i++ )
                        for ( j = 0; j < number_cols2; j++ ) {
                            int_result[ ci( i, j, number_rows ) ] = 0;
                            for ( k = 0; k < number_cols1; k++ )
                                int_result[ ci( i, j, number_rows ) ] +=
                                            int_m1[ ci( i, k, number_rows ) ] *
                                            int_m2[ ci( k, j, number_rows ) ];
                        } /*  of for  */
                      break;
        case FLOAT  : float_m1 = (float *)matrix1;
                      float_m2 = (float *)matrix2;
                      float_result = (float *)matrix3;

                      for ( i = 0; i < number_rows; i++ )
                        for ( j = 0; j < number_cols2; j++ ) {
                            int_result[ ci( i, j, number_rows ) ] = 0.0;
                            for ( k = 0; k < number_cols1; k++ )
                                float_result[ ci( i, j, number_rows ) ] +=
                                            float_m1[ ci( i, k, number_rows ) ] *
                                            float_m2[ ci( k, j, number_rows ) ];
                        } /*  of for  */
                      break;
        case DOUBLE : double_m1 = (double *)matrix1;
                      double_m2 = (double *)matrix2;
                      double_result = (double *)matrix3;
                      for ( i = 0; i < number_rows; i++ )
                        for ( j = 0; j < number_cols2; j++ ) {
                            int_result[ ci( i, j, number_rows ) ] = 0.0;
                            for ( k = 0; k < number_cols1; k++ )
                                double_result[ ci( i, j, number_rows ) ] +=
                                            double_m1[ ci( i, k, number_rows ) ] *
                                            double_m2[ ci( k, j, number_rows ) ];
                        } /*  of for  */
                      break;
        case SHORT  : short_m1 = (short *)matrix1;
                      short_m2 = (short *)matrix2;
                      short_result = (short *)matrix3;

                      for ( i = 0; i < number_rows; i++ )
                        for ( j = 0; j < number_cols2; j++ ) {
                            int_result[ ci( i, j, number_rows ) ] = 0.0;
                            for ( k = 0; k < number_cols1; k++ )
                                short_result[ ci( i, j, number_rows ) ] +=
                                            short_m1[ ci( i, k, number_rows ) ] *
                                            short_m2[ ci( k, j, number_rows ) ];
                        } /*  of for  */
                      break;
        case UNSIGN : unsigned_m1 = (unsigned int *)matrix1;
                      unsigned_m2 = (unsigned int *)matrix2;
                      unsigned_result = (unsigned int *)matrix3;

                      for ( i = 0; i < number_rows; i++ )
                        for ( j = 0; j < number_cols2; j++ ) {
                            int_result[ ci( i, j, number_rows ) ] = 0.0;
                            for ( k = 0; k < number_cols1; k++ )
                                unsigned_result[ ci( i, j, number_rows ) ] +=
                                            unsigned_m1[ ci( i, k, number_rows ) ] *
                                            unsigned_m2[ ci( k, j, number_rows ) ];
                        } /*  of for  */
                      break;
    } /*  of switch  */
} /*  of function  */
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  matrix_transpose()
*
*
* MODULE FUNCTION:
*
*  Function matrix_transpose will transpose a matrix into another matrix.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

void matrix_transpose( data_type, matrix1, number_rows, number_cols, matrix2 )

    int data_type, number_rows, number_cols;
    void *matrix1, *matrix2;

{
    register int i, j;
    int *int_m1, *int_result, row, col;
    float *float_m1, *float_result;
    double *double_m1, *double_result;
    short *short_m1, *short_result;
    unsigned int *unsigned_m1, *unsigned_result;

    switch( data_type ) {
        case INT    : int_m1 = (int *)matrix1;
                      int_result = (int *)matrix2;

                      for ( i = 0; i < number_rows; i++ )
                        for ( j = 0; j < number_cols; j++ )
                            int_result[ ci( j, i, number_rows ) ] =
                                        int_m1[ ci( i, j, number_rows ) ];
                      break;
        case FLOAT  : float_m1 = (float *)matrix1;
                      float_result = (float *)matrix2;

                      for ( i = 0; i < number_rows; i++ )
                        for ( j = 0; j < number_cols; j++ )
                            float_result[ ci( j, i, number_rows ) ] =
                                        float_m1[ ci( i, j, number_rows ) ];
                      break;
        case DOUBLE : double_m1 = (double *)matrix1;
                      double_result = (double *)matrix2;

                      for ( i = 0; i < number_rows; i++ )
                        for ( j = 0; j < number_cols; j++ )
                            double_result[ ci( j, i, number_rows ) ] =
                                        double_m1[ ci( i, j, number_rows ) ];
                      break;
        case SHORT  : short_m1 = (short *)matrix1;
                      short_result = (short *)matrix2;

                      for ( i = 0; i < number_rows; i++ )
                        for ( j = 0; j < number_cols; j++ )
                            short_result[ ci( j, i, number_rows ) ] =
                                        short_m1[ ci( i, j, number_rows ) ];
                      break;
        case UNSIGN : unsigned_m1 = (unsigned int *)matrix1;
                      unsigned_result = (unsigned int *)matrix2;

                      for ( i = 0; i < number_rows; i++ )
                        for ( j = 0; j < number_cols; j++ )
                            unsigned_result[ ci( j, i, number_rows ) ] =
                                        unsigned_m1[ ci( i, j, number_rows ) ];
                      break;
    }  /*  of switch  */
}  /*  of function  */
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  martrix_inverse()
*
*
* MODULE FUNCTION:
*
*  Function martrix_inverse will perform the inverse of a matrix.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

void matrix_inverse( data_type, matrix1, number_rows, number_cols, matrix2 )
int data_type, number_rows, number_cols;
void *matrix1, *matrix2;
{
}  /*  of function  */
```

```
#include <stdio.h>
#include <sys/stdtypes.h>
#include <sys/param.h>
#include <sys/signal.h>

#define MAX_PROCESS_NAME        256
#define NO_PROCESS_DEFINED      0
#define PROCESS_STARTED         01

struct process_data {
    char    pd_process_name[ MAX_PROCESS_NAME ];
    pid_t   pd_process_id;
    short   pd_process_status;
};

extern struct process_data process_table[];
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    status.c
*
*
* FILE FUNCTION:
*
*   This file contains the routines which build and maintain the Show Status
*   popup.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /home/project/3531/Docu/GCB.spec.doc
*
*
* FILE MODULES:
*
*   build_status_popup()     - builds the show status popup
*   cbr_show_status()        - processes the user's key presses and displays the popup
*
*****************************************<----->*****************************************/

#include <stdio.h>
#include <sys/file.h>
#include <sys/types.h>
#include <sys/time.h>
#include <dirent.h>

#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xm/Xm.h>

#include "gcb.h"
#include "cbr.h"
#include "constants.h"
#include "widgets.h"
#include "element_file.h"
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  build_status_popup()
*
* MODULE FUNCTION:
*
*   This routine builds the Show Status popup.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  build_status_popup( parent )

    Widget  parent;

{
    int x;


    dlg_dis_status = cr_popup( NULLS, parent, "Display Status" );

    FormW = cr_form( NULLS, dlg_dis_status, NULL, NULL );
    set_attribs( FORM, FormW, 650, 550, XmRESIZE_NONE );

    /*
     *  Create the status labels.
     */

    x = 3;
    cr_label("",FormW,"Position Name:",          0,x,-2,5,-2); x += 4;
    cr_label("",FormW,"Comp      Name:",         0,x,-2,5,-2); x += 4;
    cr_label("",FormW,"Element   Name:",         0,x,-2,5,-2); x += 4;
    cr_label("",FormW,"Element   Type:",         0,x,-2,5,-2); x += 4;
    cr_label("",FormW,"Current Directory:",      0,x,-2,5,-2); x += 4;
    cr_label("",FormW,"Displayer File:",         0,x,-2,5,-2); x += 4;
    cr_label("",FormW,"Error Log File:",         0,x,-2,5,-2); x += 4;
    cr_label("",FormW,"Library Path:",           0,x,-2,5,-2); x += 4;
    cr_label("",FormW,"Macros Path:",            0,x,-2,5,-2); x += 4;
    cr_label("",FormW,"Object Table:",           0,x,-2,5,-2); x += 4;
    cr_label("",FormW,"W/S Global Table:",       0,x,-2,5,-2); x += 4;
    cr_label("",FormW,"Symbol Snap:",            0,x,-2,5,-2); x += 4;
    cr_label("",FormW,"Audit:",                  0,x,-2,5,-2); x += 4;
    cr_label("",FormW,"Target Language:",        0,x,-2,5,-2); x += 4;
    cr_label("",FormW,"User Functions Path:",    0,x,-2,5,-2); x += 4;
    cr_label("",FormW,"User Name:",              0,x,-2,5,-2); x += 12;
    cr_label("",FormW,"Time",                    0,x,-2,5,-2); x += 4;
    cr_label("",FormW,"Date",                    0,x,-2,5,-2);

    x = 3;
    lbl_val_pname    = cr_label("",FormW,"",0,x,-2,30,-2); x += 4;
    lbl_val_cname    = cr_label("",FormW,"",0,x,-2,30,-2); x += 4;
    lbl_val_ename    = cr_label("",FormW,"",0,x,-2,30,-2); x += 4;
    lbl_val_type     = cr_label("",FormW,"",0,x,-2,30,-2); x += 4;
    lbl_val_cwd      = cr_label("",FormW,"",0,x,-2,30,-2); x += 4;
    lbl_val_cflow    = cr_label("",FormW,"",0,x,-2,30,-2); x += 4;
    lbl_val_err_file = cr_label("",FormW,"",0,x,-2,30,-2); x += 4;
    lbl_val_lib_path = cr_label("",FormW,"",0,x,-2,30,-2); x += 4;
    lbl_val_mac_path = cr_label("",FormW,"",0,x,-2,30,-2); x += 4;
```

```
    lbl_val_msid     = cr_label("",FormW,"",0,x,-2,30,-2); x += 4;
    lbl_val_ws_glob  = cr_label("",FormW,"",0,x,-2,30,-2); x += 4;
    lbl_val_snap     = cr_label("",FormW,"",0,x,-2,30,-2); x += 4;
    lbl_val_audit    = cr_label("",FormW,"",0,x,-2,30,-2); x += 4;
    lbl_val_language = cr_label("",FormW,"",0,x,-2,30,-2); x += 4;
    lbl_val_func_path= cr_label("",FormW,"",0,x,-2,30,-2); x += 4;
    lbl_val_user     = cr_label("",FormW,"",0,x,-2,30,-2); x += 12;
    lbl_val_time     = cr_label("",FormW,"",0,x,-2,30,-2); x += 4;
    lbl_val_date     = cr_label("",FormW,"",0,x,-2,30,-2);

    cr_separator( NULLS, FormW, 90, IGNORE, 2, 98 );

    CancelW  = cr_command( NULLS, FormW, "Close", cbr_show_status, CANCEL );
    HelpW    = cr_command( NULLS, FormW, "Help",  cbr_help,        STATUS_HELP);

    set_position ( CancelW, 95, IGNORE, 15, IGNORE);
    set_position ( HelpW,   95, IGNORE, 70, IGNORE);
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_show_status()
*
* MODULE FUNCTION:
*
*   This routine pops up the Show Status popup.  This routine also processes the
*   user's key presses in the popup.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_show_status( w, client_data, call_data )

    Widget  w;
    int     client_data;
    caddr_t call_data;

{
    char    dstr[40],
            tstr[40];
    time_t  clock;

    if (Mode != EditSymbol)
        return;

    /*
     *  User wants to abort from viewing the status popup, take down the popup.
     */

    if ( client_data == CANCEL )
        {
        XtUnmanageChild( dlg_dis_status );
        return;
        }

    /*
     *  Get the time and date from the system to display to the user.
     */

    clock = time( NULL );
    strftime( dstr, 39, "%a,  %h %e, %Y", localtime(&clock) );
    strftime( tstr, 9,  "%T",             localtime(&clock) );
    set_label( lbl_val_date, dstr );
    set_label( lbl_val_time, tstr );

    /*
     *  Load and show the Status popup.
     */

    load_curr_dir( Cwd );
    set_label( lbl_val_cwd, Cwd );

    if ( Snap )
        set_label( lbl_val_snap, "On" );
    else
        set_label( lbl_val_snap, "Off" );
```

```
    if ( Audit )
        set_label( lbl_val_audit, "On" );
    else
        set_label( lbl_val_audit, "Off" );

    switch ( ElementType )
        {
        case ELEMENT : set_label( lbl_val_type, "Comp Element" );       break;
        case LIB     : set_label( lbl_val_type, "Library Element" );    break;
        default      : set_label( lbl_val_type, NULLS );                break;
        }

    switch ( TargetLanguage )
        {
        case C    : set_label( lbl_val_language, "C"    ); break;
        case MOAL : set_label( lbl_val_language, "MOAL" ); break;
        case UIL  : set_label( lbl_val_language, "UIL"  ); break;
        }

    set_label( lbl_val_pname,       pPosition   );
    set_label( lbl_val_cflow,       DisplayFile );
    set_label( lbl_val_cname,       CompFile    );
    set_label( lbl_val_ename,       ElementFile );
    set_label( lbl_val_err_file,    LogFile     );
    set_label( lbl_val_func_path,   UserFuncsPath );
    set_label( lbl_val_lib_path,    LibPath     );
    set_label( lbl_val_mac_path,    MacrosPath );
    set_label( lbl_val_msid,        MSIDTable  );
    set_label( lbl_val_ws_glob,     WSGlobals  );
    set_label( lbl_val_user,        UserName   );

    XtManageChild( dlg_dis_status );
}
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    sub_menus.c
*
* FILE FUNCTION:
*
*   This file contains the routines which create the various Expression menu submenus:
*
*       Matrix submenu
*       Trig   submenu
*
*   This file also contains the routines which process these menus.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /home/project/3531/Docu/GCB.spec.doc
*
*
* FILE MODULES:
*
*   cbr_matrix()        - process the user's button presses in the Matrix menu
*   cbr_quaternion()    - process the user's button presses in the Quaternion menu
*   cbr_trig()          - process the user's button presses in the Trig menu
*   setup_matrix_menu() - build the Matrix menu
*   setup_trig_menu()   - build the Trig menu
*
*****************************************<----->*****************************************/

#include <stdio.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xm/Xm.h>
#include <Xm/Form.h>

#include "gcb.h"
#include "widgets.h"
#include "constants.h"
#include "menu.h"
#include "symbol.h"
#include "gcb_parse.h"
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_matrix()
*
* MODULE FUNCTION:
*
*   This routine performs two functions:
*
*       1) display the Matrix menu when the user picks Matrix op during expressions
*       2) add the matrix op to the expression when one is selected
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_matrix( w, client_data, call_data )

    Widget      w;
    caddr_t     client_data,
                call_data;

{
    XmTextPosition pos = XmTextGetInsertionPosition( scr_expr );

    switch ( (int) client_data )
        {
        case M_SHOW :

            /*
             *  Take down the math/logical elements frame and put up the matrix frame.
             *  Set the Mode to MatrixMenu so the CANCEL button will know to restore
             *  the math frame.  Make sure we "return()" because there is code past
             *  the "switch" which only relates to adding to the expression.
             */

            XtUnmapWidget( frame_math_menu );
            XtMapWidget( frame_matrix_menu );
            OldMode = Mode;
            Mode = MatrixMenu;
            return;

        case M_ADD    : XmTextInsert( scr_expr, pos, " ADD"     ); break;
        case M_SUB    : XmTextInsert( scr_expr, pos, " SUB"     ); break;
        case M_MULT   : XmTextInsert( scr_expr, pos, " MULT"    ); break;
        case M_IDENT  : XmTextInsert( scr_expr, pos, " IDENT"   ); break;
        case M_INV    : XmTextInsert( scr_expr, pos, " INVERSE" ); break;
        case M_TRAN   : XmTextInsert( scr_expr, pos, " TRANSP"  ); break;
        case M_CROSS  : XmTextInsert( scr_expr, pos, " CROSS"   ); break;
        case M_DOT    : XmTextInsert( scr_expr, pos, " DOT"     ); break;
        }

    /*
     *  Now take down the matrix menu, and parse the expression.
     */

    XtUnmapWidget( frame_matrix_menu );
    XtMapWidget( frame_math_menu );
    Mode = OldMode;
    do_parse();
```

```
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_quaternion()
*
* MODULE FUNCTION:
*
*   This routine is just a stub routine for now.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0 - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_quaternion( w, client_data, call_data )

    Widget      w;
    caddr_t     client_data,
                call_data;

{
    user_ack("Sorry, the Quaternion functions are yet implemented in the GCB");
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  cbr_trig()
*
* MODULE FUNCTION:
*
*   This routine performs two functions:
*
*       1) display the Trig menu when the user picks Trig op during expressions
*       2) add the Trig op to the expression when one is selected
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

XtCallbackProc  cbr_trig( w, client_data, call_data )

    Widget      w;
    caddr_t     client_data,
                call_data;

{
    XmTextPosition pos = XmTextGetInsertionPosition( scr_expr );

    switch ( (int) client_data )
        {
        case T_SHOW :

            /*
             *  Take down the math/logical elements frame and put up the trig frame.
             *  Set the Mode to TrigMenu so the CANCEL button will know to restore
             *  the math frame.  Make sure we "return()" because there is code past
             *  the "switch" which only relates to adding to the expression.
             */

            XtUnmapWidget( frame_math_menu );
            XtMapWidget( frame_trig_menu );
            OldMode = Mode;
            Mode = TrigMenu;
            return;

        case T_COS  : XmTextInsert( scr_expr, pos, " cos"  ); break;
        case T_ACOS : XmTextInsert( scr_expr, pos, " acos" ); break;
        case T_TAN  : XmTextInsert( scr_expr, pos, " tan"  ); break;
        case T_ATAN : XmTextInsert( scr_expr, pos, " atan" ); break;
        case T_SIN  : XmTextInsert( scr_expr, pos, " sin"  ); break;
        case T_ASIN : XmTextInsert( scr_expr, pos, " asin" ); break;
        }

    /*
     *  Now take down the trig menu, and parse the expression.
     */

    XtUnmapWidget( frame_trig_menu );
    XtMapWidget( frame_math_menu );
    Mode = OldMode;
    do_parse();
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  setup_matrix_menu()
*
* MODULE FUNCTION:
*
*   This routine builds the Matrix menu.  The Matrix menu overlays the logical
*   expression menu when the user chooses to add a matrix operation to an expression.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  setup_matrix_menu( parent, under )

    Widget parent, under;

{
    Arg     args[5];
    Widget  w[8];
    int     n;

    /*
     *  create frame to hold matrix menu widgets
     */

    frame_matrix_menu = cr_frame( NULLS, parent, NULL, under);
    XtSetArg( args[0], XtNmappedWhenManaged, FALSE );
    XtSetValues( frame_matrix_menu, args, 1 );

    /*
     *  create form to hold rowcols and labels
     */

    n = 0;
    XtSetArg( args[n], XmNwidth,  278 ); n++;
    XtSetArg( args[n], XmNheight, 464 ); n++;
    frm_matrix_menu = XmCreateForm( frame_matrix_menu, NULLS, args, n);
    XtManageChild( frm_matrix_menu );

    /*
     *  create labels in matrix menu
     */

    cr_label( NULLS, frm_matrix_menu, "GCB Elements",     1,  1,  7, 1, 97 );
    cr_label( NULLS, frm_matrix_menu, "Matrix Functions",0, 25, 28, 1, 99 );

    /*
     *  create rowcol to hold matrix operation buttons
     */

    cr_separator( NULLS, frm_matrix_menu, 29, 30, 1, 97 );
    rc_mat = cr_rowcol( NULLS, frm_matrix_menu, 1, XmVERTICAL, NULL, NULL);
    set_position( rc_mat, 33, IGNORE, 25, IGNORE);

    w[0] = cr_command( NULLS, rc_mat, "       Add        ", cbr_matrix, M_ADD   );
    w[1] = cr_command( NULLS, rc_mat, "     Subtract     ", cbr_matrix, M_SUB   );
    w[2] = cr_command( NULLS, rc_mat, "     Multiply     ", cbr_matrix, M_MULT  );
    w[3] = cr_command( NULLS, rc_mat, "     Identity     ", cbr_matrix, M_IDENT );
    w[4] = cr_command( NULLS, rc_mat, "      Inverse     ", cbr_matrix, M_INV   );
    w[5] = cr_command( NULLS, rc_mat, "     Transpose    ", cbr_matrix, M_TRAN  );
    w[6] = cr_command( NULLS, rc_mat, "  Cross Product   ", cbr_matrix, M_CROSS );
    w[7] = cr_command( NULLS, rc_mat, "  Dot   Product   ", cbr_matrix, M_DOT   );

    /*
     *  Set the state of the buttons to "active".
     */

    for ( n=0; n<8; n++ )
        {
        XtSetArg( args[0], XmNbackground,
            (XtArgVal) WhitePixel(display, DefaultScreen(display)) );
        XtSetValues( w[n], args, 1 );
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  setup_trig_menu()
*   ||
* MODULE FUNCTION:
*
*   This routine builds the Trigonometric menu.  The Trigonometric menu overlays the
*   logical expression menu when the user chooses to add a Trig operation to an
*   expression.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  setup_trig_menu( parent, under )

    Widget parent, under;

{
    Arg     args[5];
    Widget  w[6];
    int     n;

    /*
     *  create frame to hold trig menu widgets
     */

    frame_trig_menu = cr_frame( NULLS, parent, NULL, under);
    XtSetArg( args[0], XtNmappedWhenManaged, FALSE );
    XtSetValues( frame_trig_menu, args, 1 );

    /*
     *  create form to hold rowcols and labels
     */

    n = 0;
    XtSetArg( args[n], XmNwidth,  278 ); n++;
    XtSetArg( args[n], XmNheight, 464 ); n++;
    frm_trig_menu = XmCreateForm( frame_trig_menu, NULLS, args, n);
    XtManageChild( frm_trig_menu );

    /*
     *  create labels in trig menu
     */

    cr_label( NULLS, frm_trig_menu, "GCB Elements",           1,  1,  7, 1, 97 );
    cr_label( NULLS, frm_trig_menu, "Trigonometric Functions",0, 25, 28, 1, 99 );

    /*
     *  create rowcol to hold trig operation buttons
     */

    cr_separator( NULLS, frm_trig_menu, 29, 30, 1, 97 );
    rc_trig = cr_rowcol( NULLS, frm_trig_menu, 1, XmVERTICAL, NULL, NULL);
    set_position( rc_trig, 33, IGNORE, 25, IGNORE);

    w[0] = cr_command( NULLS, rc_trig, "        Cosine        ", cbr_trig, T_COS  );
    w[1] = cr_command( NULLS, rc_trig, "     Arc   Cosine     ", cbr_trig, T_ACOS );
    w[2] = cr_command( NULLS, rc_trig, "        Tangent       ", cbr_trig, T_TAN  );
    w[3] = cr_command( NULLS, rc_trig, "     Arc Tangent      ", cbr_trig, T_ATAN );
    w[4] = cr_command( NULLS, rc_trig, "         Sine         ", cbr_trig, T_SIN  );
    w[5] = cr_command( NULLS, rc_trig, "      Arc  Sine       ", cbr_trig, T_ASIN );

    /*
     *  Set the state of the buttons to "active".
     */

    for ( n=0; n<6; n++ )
        {
        XtSetArg( args[0], XmNbackground,
            (XtArgVal) WhitePixel(display, DefaultScreen(display)) );
        XtSetValues( w[n], args, 1 );
        }
}
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    symbol.h
*
* FILE FUNCTION:
*
*   This file contains the constants and data strucutres used to build the GCB
*   symbol tables.  The symbol table is implemented as a linked list of symbols.
*   Procedure labels are entries on the primary linked list with subordinate
*   symbol table lists within each procedure.
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

#define MAX_SYMBOL_SIZE     128

/*
 *  Constants for symbol table entries.  Octal bit masks for symbol table
 *  attributes.
 *
 *  Constants for data types of either variables or constants.
 */

#define SYMBOL_TYPE         077

#define VARIABLE            01
#define LOCAL_VAR           02
#define GLOBAL_VAR          04

#define PROCEDURE           010
#define WS_GLOBAL           020
#define WS_OBJECT           040

#define CHAR                0100
#define SHORT               0200
#define INTEGER             0400

#define UNSIGNED            01000
#define FLOAT               02000
#define DOUBLE              04000

#define INTRINSIC           0100000
#define INSTALLED           0200000
#define COMP_REF            0400000
#define NOT_INSTALLED       0577777

#define LOOKUP_ONLY         0
#define SYMBOL_DATA_TYPE    07700
#define SYMBOL_ATTRIBUTES   077700
#define MATRIX              010000

#define MAX_DIMENSIONS      4
#define SUB1                0
#define SUB2                1
#define SUB3                2
#define SUB4                3

struct symbol_entry {
    int                 se_type;
    short               se_num_dimensions;
    short               se_subs[ MAX_DIMENSIONS ];
    int                 se_use_count;
    char                se_symbol[ MAX_SYMBOL_SIZE + 1 ];
    struct symbol_entry *se_next;
    struct symbol_entry *se_local_vars;
};  /*  of structure  */

/*
 *  The following global variable serves as the root of the symbol table.  This variable
 *  is initialized when a new comp is initiated.
 */

extern struct symbol_entry *symbol_table;

/*
 *  Various error codes.
 */

#define NO_MEMORY           -1
#define UNKNOWN_PARENT      -2
#define DUPLICATE_ENTRY     -3

/*
 *  Function prototypes.
 */

struct symbol_entry *lookup_symbol();

void                remove_symbol_entry();

int                 copy_element_to_symbol_table(),
                    del_elem_from_symbol_table();
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    symbol_table.c
*
* FILE FUNCTION:
*
*   This file contains the routines which add to and lookup in the symbol table.
*
*   The symbol table is stored as a linked list which will contain all entry points
*   for a given COMP.  The symbol table is structured as a single rooted, linked list
*   with all global entry points defined in this root list; any local variables; those
*   variables defined within an element, are stored in a similarly structured link list
*   which is connected to the root linked list through a pointer.  This structure allows
*   the symbol table routines to be recursive in nature since each level of the symbol
*   table utilizes identical C structures.  Although the GCB does not allow nested
*   elements, the symbol table structure defined and implemented would allow nesting.
*
*
* FILE MODULES:
*
*   add_symbol_entry            ()  - add a symbol entry to the symbol table
*   copy_element_to_symbol_table()  - add a new element to the symbol table
*   del_elem_from_symbol_table  ()  - delete an element from the symbol table
*   decrement_symbol_use_count  ()  - decrement a symbol's use count
*   increment_symbol_use_count  ()  - increment a symbol's use count
*   lookup_symbol               ()  - determine if a symbol is in the symbol table
*   lookup_local_varlist        ()  - retrieve the local variable list for a procedure
*   print_symbol_table          ()  - print the contents of the symbol table
*   remove_symbol_entry         ()  - remove an entry from the symbol table
*   restore_symbol_level        ()  - read a symbol table entry from disk
*   return_symbol_type          ()  - return the type (e.g. variable) of a symbol
*   return_matrix_attributes    ()  - return the attributes (e.g. data type) of a matrix
*   return_symbol_attributes    ()  - return the attributes (e.g. data type) of a symbol
*   set_sym_attribs             ()  - sets the symbol table attributes of an entry
*   symbol_table_init           ()  - initialize the symbol table to default values
*
*****************************************<----->*****************************************/

#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>

#include "gcb.h"
#include "symbol.h"
#include "var_list.h"

/*
 *  The following global variable serves as the root of the symbol table.  The variable
 *  is initialized when a new comp is initiated.
 */

struct symbol_entry *symbol_table;
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  copy_element_to_symbol_table()
*
* MODULE FUNCTION:
*
*   This routine is called in element_file.c to copy an existing Element's symbol
*   table information to the symbol table for a new Element's name.  This routine
*   is called when the user copies an existing Element to another Element.  This
*   routine will make sure the local variables and global variable use counts are
*   properly updated for the new element.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.01 - 08/01/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  copy_element_to_symbol_table( OldElement )

    char *OldElement;

{
    struct symbol_entry *new,           /* ptr to new Element name symbol entry     */
                        *newEntry,      /* ptr to new local variable symbol entry   */
                        *oldEntry,      /* ptr to org local variable symbol entry   */
                        *org;           /* ptr to original Element name symbol entry*/
    struct list_elem    *tlist,
                        *vlist;
    int sym;


    /*
     *  See if the new Element name is in the symbol table, if it is not, add the new
     *  Element name to the symbol table.
     */

    if (! (struct symbol_entry *) lookup_symbol(NULL,ElementFile) )
        if ( add_symbol_entry(NULL,ElementFile,PROCEDURE,0,0) )
            error_handler( ERR_ADD_SYMBOL, "copy_element_to_symbol_table" );

    /*
     *  Locate the original Element name in the symbol table.
     */

    if (! (org = lookup_symbol(NULL,OldElement)) )
        {
        elog(1,"Copy Element: couldn't find original element in symbol table");
        return( ERR );
        }

    /*
     *  Traverse the list of local variables and add them to the new Element's
     *  list of local variables.
     */

    org = org->se_local_vars;
    while ( org != NULL )
        {
```

```
        /*
         *  Add a new entry into the symbol table for the current local variable
         *  from the original Element.
         */

        if ( add_symbol_entry(ElementFile,
                              org->se_symbol,
                              org->se_type,
                              org->se_num_dimensions,
                              org->se_subs[ SUB1 ],
                              org->se_subs[ SUB2 ],
                              org->se_subs[ SUB3 ],
                              org->se_subs[ SUB4 ]) )
            error_handler( ERR_ADD_SYMBOL, "copy_element_to_symbol_table" );

        /*
         *  Set use count to the original entry's use count.  Locate the original
         *  entry and then locate the new entry.  Then set the new entry's use count
         *  to the original's use count.
         */

        if (! (oldEntry = lookup_symbol(OldElement,org->se_symbol)) )
            {
            elog(1,"Copy Element: couldn't find original local var in symbol table");
            return( ERR );
            }

        if (! (newEntry = lookup_symbol(ElementFile,org->se_symbol)) )
            {
            elog(1,"Copy Element: couldn't find new local variable in symbol table");
            return( ERR );
            }

        newEntry->se_use_count = oldEntry->se_use_count;

        /*
         *  Get a pointer to the next local variable in the list.
         */

        org = org->se_next;
        }

    /*
     *  Loop through each of the symbols locating references to global variables.
     *  Increment the use count of each global variable we encounter.
     */

    for ( sym=0; sym<MAX_SYMBOLS; sym++ )
        {

        /*
         *  Ignore empty symbols.
         */

        if ( Symbol_Map[sym].symbol_type == NONE )
            continue;

        /*
         *  If the symbol is a GOTO, START, or STOP, increment the use count of the
         *  Element/Comp name.
         */

        if ( (Symbol_Map[sym].symbol_type == GOTO) ||
             (Symbol_Map[sym].symbol_type == STOP) ||
             (Symbol_Map[sym].symbol_type == START) )
            increment_symbol_use_count( NULL, Symbol_Map[sym].text );

        /*
         *  If the symbol is an IF or SET, check the expression for global variable
         *  references.
         */

        else if ( (Symbol_Map[sym].symbol_type == IF) ||
                  (Symbol_Map[sym].symbol_type == SET))
            {
            tlist = vlist = build_var_list( Symbol_Map[sym].Sym.IfSym.comp_expr,
                                            ElementFile );

            while ( tlist )
                {
                if ( (is_local(tlist->var_name) & GLOBAL_VAR) )
                    increment_symbol_use_count( NULL, tlist->var_name );
                tlist = tlist->next;
                }

            if ( vlist )
                destroy_var_list( vlist );
            }
        }

    return( OK );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  del_elem_from_symbol_table()
*
* MODULE FUNCTION:
*
*   This routine is called to delete an Element from the Symbol table.  The
*   specified Element file is opened up and all of the expressions are located
*   within the Element so that any references to global variables can be
*   decremented in the Symbol table.  Also, any CALLS, ACTIVATES, or STOPS
*   are located so that the use count of the Element names can be updated.
*   Once the symbol table has been updated, then the Element and its local
*   variables are removed from the symbol table.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                                Release 1.01 - 08/01/91
*                                Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  del_elem_from_symbol_table( el_file, gel_file )

    char     *el_file,      /* element name without path or extension   */
            *gel_file;      /* element name, path and extension         */

{
    FILE    *fp;
    char    elementName[MAX_NAME],
            expr[MAX_TEXT];
    int     count = 0,
            rc,
            symbol_type;
    struct symbol_entry *symEntry;
    struct list_elem    *tlist,
                        *vlist;


    /*
     *  Open the Element file to be deleted and locate any global variable references
     *  so we can decrement their use counts.
     */

    if (! (fp = fopen(gel_file,"r")) )
        {
        elog(3,"Couldn't open Element file in del_elem_from_symbol_table(): %s",
                gel_file);

        /*
         *  Couldn't open file, remove the Element name and all of its local variables
         *  from the symbol table.
         */

        if (! (symEntry = lookup_symbol(NULL,el_file)) )
            elog(1,"Delete Element: couldn't find Element name in the symbol table");
        else
            remove_symbol_entry( NULL, symEntry );

        return( ERR );
        }
```

```
    /*
     *  Locate a graphical symbol which will contain an expression or Element file
     *  name.
     */

    while ( (rc = fscanf(fp,"%d",&symbol_type)) != EOF )
        {

        /*
         *  If we don't get an integer, dump the rest of the line.  If we
         *  get a LINE SEGMENT indicator, we are finished with the file
         *  because the line info follows the symbol info.
         */

        if ( rc == 0 )
            {
            fscanf( fp, "%*[^\n]" );
            continue;
            }

        if ( symbol_type == SEGMENT_KEY )
            break;

        /*
         *  See if we have a symbol which may contain a reference to an
         *  element file or may contain references to variables.  If we find
         *  a symbol with a reference to an Element, decrement the use count
         *  of the Element name.  If we find a symbol with an expression, build
         *  list of the variables in the expression, and then decrement the use
         *  count of the global variables in the expression.
         */

        switch ( symbol_type )
            {
            case START:
            case STOP:
            case GOTO:

                fscanf( fp,"%*u %*d %*d %*d %*d %*d %*d %*d %*d %*d" );
                read_element_str( fp, elementName );
                fscanf( fp, "%*[^\n]" );

                /*DEBUG*/
                elog(3,"del_sym_tab: elementName: %s",elementName);

                decrement_symbol_use_count( NULL, elementName );

                break;

            case SET:
            case IF:

                fscanf( fp,"%*u %*d %*d %*d %*d %*d %*d %*d %*d %*d" );
                read_element_str( fp, NULL );
                fscanf( fp,"%*u %*u" );

                if ( symbol_type == IF )
                    fscanf( fp,"%*u %*u %*d %*d %*d" );

                /*
                 *  Read the symbol's comp expression.
                 */

                read_element_str( fp, NULL );
```

```
            read_element_str( fp, expr );
            fscanf( fp, "%*[^\n]" );

            /*DEBUG*/
            elog(3,"del_sym_tab: expression: %s",expr);

            /*
             *  Now build a list of the variables in the expression.
             */

            tlist = vlist = build_var_list( expr, el_file );

            /*
             *  Loop through the list of variables and decrement the use count
             *  of each global variable.
             */

            while ( tlist )
                {
                if ( (is_local(tlist->var_name) & GLOBAL_VAR) )
                    decrement_symbol_use_count( NULL, tlist->var_name );
                tlist = tlist->next;
                }

            /*
             *  Free() the list of variables.
             */

            if ( vlist )
                destroy_var_list( vlist );
            break;

        default :

            fscanf( fp, "%*[^\n]" );
            continue;
        }
    }

    /*
     *  Close the element file.
     */

    fclose( fp );

    /*
     *  Remove the Element name and all of its local variables from the symbol
     *  table.
     */

    if (! (symEntry = lookup_symbol(NULL,el_file)) )
        {
        elog(1,"Delete Element: couldn't find Element name in the symbol table");
        return( ERR );
        }

    remove_symbol_entry( NULL, symEntry );

    return( OK );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  lookup_symbol()
*
*
* MODULE FUNCTION:
*
*  Function lookup_symbol will search the symbol table for the specified entry and
*  return a pointer to the entry if found.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

struct symbol_entry  *lookup_symbol( parent, name )

    char *parent, *name;

{
    register int not_done;
    struct symbol_entry *temp;

    /*
     *  Initialize temp to point to the root of the symbol table.
     */

    temp = symbol_table;

    /*
     *  If a parent was specified, first find the procedure entry in the symbol table.
     */

    if ( parent != NULL ) {

        /*
         *  Scan down the symbol table looking first for a PROCEDURE entry and then
         *  determine if the symbol can be found.
         */

        not_done = 1;
        while ( not_done && temp )
            if ( ( temp->se_type & PROCEDURE ) &&
                 ( strcmp( temp->se_symbol, parent ) == 0 ) ) {
                temp = temp->se_local_vars;
                not_done = 0;
            }
            else
                temp = temp->se_next;

        /*
         *  If temp == NULL then that implies the specified parent was not found.  If
         *  this is the case then NULL should be returned.
         */

        if ( temp == NULL )
            return( NULL );
    }

    /*
```

```
     *  Scan the symbol list looking for a match.  If the match is found return a
     *  pointer to the entry.
     */

    not_done = 1;
    while ( not_done && temp )
        if ( strcmp( temp->se_symbol, name ) == 0 )
            return( temp );
        else
            temp = temp->se_next;

    /*
     *  If the symbol had a parent specified and no match was found in the local
     *  variable list of the procedure, determine if the symbol can be resolved as
     *  a global variable.
     */

    if ( parent != NULL ) {
        temp = symbol_table;
        not_done = 1;
        while ( not_done && temp )
            if ( strcmp( temp->se_symbol, name ) == 0 )
                return( temp );
            else
                temp = temp->se_next;
    }

    /*
     *  Return a NULL as no match was found.
     */

    return( NULL );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  lookup_local_varlist()
*
*
* MODULE FUNCTION:
*
*  Function lookup_local_varlist will search the symbol table for the specified parent,
*  when found, the function will return a pointer to the list of local variables.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

struct symbol_entry *lookup_local_varlist( parent )

    char *parent;

{
    struct symbol_entry *temp;

    /*
     *  Initialize temp to point to the root of the symbol table.
     */

    temp = symbol_table;

    /*
     *  If a parent was specified, first find the procedure entry in the symbol table.
     */

    if ( parent )

        /*
         *  Scan down the symbol table looking first for a PROCEDURE entry and then
         *  determine if the symbol can be found.
         */

        while ( temp )
            if ( ( temp->se_type & PROCEDURE ) &&
                 ( strcmp( temp->se_symbol, parent ) == 0 ) )
                return( temp->se_local_vars );
            else
                temp = temp->se_next;

    /*
     *  Either an invalid parent or no parent was specified.  Return NULL.
     */

    return( NULL );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  add_symbol_entry()
*
*
* MODULE FUNCTION:
*
*  Function add_symbol_entry will add an entry to the symbol table.  The function
*  expects three arguments:
*
*       PARENT: The name of the enclosing logical block (i.e. NULL for a global
*               variable). If the variable is local to a procedure then the PARENT
*               will be the procedure name where the declaration occurred.
*
*       NAME:   The name of the symbol to be entered.
*
*       ATTRIBUTES: The attributes of the symbol table entry to be made.
*
*  The function will return the following:
*
*        0:  Success
*       -1:  No memory
*       -2:  Unknown parent
*       -3:  Duplicate entry
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  add_symbol_entry( parent, name, attributes, num_dimensions, sub1, sub2, sub3, sub4
)

    char    *parent, *name;
    int     attributes;
    short   num_dimensions, sub1, sub2, sub3, sub4;

{
    register int not_done;
    struct symbol_entry *temp, *symbol_head;

    /*
     *  If the symbol exists then do not allow it to be added.
     */

    if ( lookup_symbol( parent, name ) )
        return( DUPLICATE_ENTRY );

    /*
     *  Allocate space for the symbol and establish all default (initial values).
     */

    if ( temp = (struct symbol_entry *)malloc( sizeof( struct symbol_entry ) ) ) {
        strcpy( temp->se_symbol, name );
        temp->se_type = attributes;
        temp->se_use_count = 0;
        temp->se_num_dimensions = num_dimensions;
        temp->se_subs[ SUB1 ] = sub1;
        temp->se_subs[ SUB2 ] = sub2;
        temp->se_subs[ SUB3 ] = sub3;
        temp->se_subs[ SUB4 ] = sub4;
        temp->se_next = NULL;
        temp->se_local_vars = NULL;

        /*
         *  Set the LOCAL/GLOBAL bit based on whether or not a parent exists.  If the
         *  variable is a global variable, determine if it is either a workstation
         *  global or an object.
         */

        if ( attributes & VARIABLE )  {

            /*
             *  Determine if the variable is a global variable.
             */

            if ( parent == NULL ) {
                temp->se_type |= GLOBAL_VAR;

                /*
                 *  Determine if the variable is either a workstation global or an
                 *  object.
                 */

                if ( strncmp( name, "WS_", 3 ) == 0 )
                    temp->se_type |= WS_GLOBAL;
                else if ( name[0] == 'V' )
                    temp->se_type |= WS_OBJECT;
            }
            else
                temp->se_type |= LOCAL_VAR;
        }
    }
    else
        return( NO_MEMORY );

    /*
     *  If the entry is the first symbol in the symbol table, then handle the special
     *  case and return.
     */

    if ( symbol_table == NULL ) {
        symbol_table = temp;
        return( SUCCESS );
    }

    /*
     *  The symbol needs to be added to the symbol table linked list.  If a parent was
     *  specified then find the local variable symbol list.
     */

    symbol_head = symbol_table;
    if ( parent != NULL ) {
        not_done = 1;
        while ( not_done && symbol_head )
            if ( ( symbol_head->se_type & PROCEDURE ) &&
                 ( strcmp( symbol_head->se_symbol, parent ) == 0 ) ) {
                not_done = 0;

                /*
                 *  If the symbol to be added is the first local variable, then add it
                 *  to the list and return, otherwise set the local variable so that the
                 *  list can be traversed.
                 */
```

```
            if ( symbol_head->se_local_vars == NULL ) {
                symbol_head->se_local_vars = temp;
                return( SUCCESS );
            }
            else
                symbol_head = symbol_head->se_local_vars;
        }
        else
            symbol_head = symbol_head->se_next;

    if ( symbol_head == NULL )
        return( UNKNOWN_PARENT );
}

/*
 *  The variable symbol_head now points to the head of the symbol list, traverse to
 *  the end of the list and add the newlw allocated symbol.
 */

while ( symbol_head->se_next )
    symbol_head = symbol_head->se_next;
symbol_head->se_next = temp;

/*
 *  Issue a successful error code.
 */

return( SUCCESS );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  return_symbol_type()
*
*
* MODULE FUNCTION:
*
*  Function return_symbol_type will return the symbol type.  A -1 is returned if the
*  symbol cannot be found.
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  return_symbol_type( parent, name )

    char *parent, *name;

{
    struct symbol_entry *temp;

    /*
     *  Fetch a pointer to the symbol entry.  If NULL is returned then the symbol was
     *  not found ... a -1 should be returned.
     */

    if ( temp = lookup_symbol( parent, name ) )
        return( temp->se_type & SYMBOL_TYPE );
    else
        return( ERR );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  return_symbol_attributes()
*
*
* MODULE FUNCTION:
*
*  Function return_symbol_attributes will return the symbol attributes (if it is not a
*  procedure).
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  return_symbol_attributes( parent, name )

    char *parent, *name;

{
    struct symbol_entry *temp;

    /*
     *  Fetch a pointer to the symbol entry.  If NULL is returned then the symbol was no
t
     *  found ... a -1 should be returned.
     */

    if ( ( temp = lookup_symbol( parent, name ) ) &&
         ( !(temp->se_type & PROCEDURE ) ) )
        return( temp->se_type & SYMBOL_ATTRIBUTES );
    else
        return( ERR );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  return_matrix_attributes()
*
*
* MODULE FUNCTION:
*
*  Function return_matrix_attributes will return the number of rows and columns associate
d
*  with the MATRIX variable.
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  return_matrix_attributes( parent, name, dimensions, sub1, sub2, sub3, sub4 )

    char *parent, *name;
    short *dimensions, *sub1, *sub2, *sub3, *sub4;

{
    struct symbol_entry *temp;

    /*
     *  Fetch a pointer to the symbol entry.  If NULL is returned then the symbol was not
     *  found ... a -1 should be returned.
     */

    if ( ( temp = lookup_symbol( parent, name ) ) && ( temp->se_type & MATRIX ) ) {
        *dimensions = temp->se_num_dimensions;
        *sub1 = temp->se_subs[ SUB1 ];
        *sub2 = temp->se_subs[ SUB2 ];
        *sub3 = temp->se_subs[ SUB3 ];
        *sub4 = temp->se_subs[ SUB4 ];
        return( 0 );
    }
    else
        return( ERR );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  increment_symbol_use_count()
*
*
* MODULE FUNCTION:
*
*  Function increment_symbol_use_count will increment the use count of a symbol.  This
*  function exists so that symbols can be deleted.
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  increment_symbol_use_count( parent, name )

    char *parent, *name;

{
    struct symbol_entry *temp;

    /*
     *  Lookup the symbol, if found, increment the use count.
     */

    if ( temp = lookup_symbol( parent, name ) )
        return( temp->se_use_count++ );
    else
        return( ERR );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  remove_symbol_entry()
*
*
* MODULE FUNCTION:
*
*   Function remove_symbol_entry is invoked to remove a symbol from the symbol table
*   (which occurs when the symbol's use count goes to zero).  This routine will also
*   remove an Element's local variable list automatically when the Element is to be
*   deleted.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.01 - 08/01/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

void  remove_symbol_entry( parent, entry_to_delete )

    char *parent;
    struct symbol_entry *entry_to_delete;

{
    struct symbol_entry *temp,
                        *temp1,
                        *head;

    /*
     *  If the entry to be deleted is not in the root of the symbol table (which can
     *  be determined if the parent is not NULL), find the list which is owned by the
     *  parent.
     */

    head = symbol_table;
    if ( parent != NULL ) {

        /*
         *  Scan down the symbol table looking first for a match to the parent.  The
         *  loop will not fail as this routine would not have been invoked if the
         *  parent did not exist in the symbol table.
         */

        temp = symbol_table;
        while ( strcmp( temp->se_symbol, parent ) != 0 )
            temp = temp->se_next;

        /*
         *  Establish the pointer to the head of the list.
         */

        head = temp->se_local_vars;
    }

    /*
     *  Now that the list and the entry are established, delete the entry.  If the
     *  pointer is to the header of a local list of variables (i.e. not a pointer to
     *  the head of the symbol table), the pointer to the beginning of the local
     *  variable list is update.
```

```
     */

    if ( head == entry_to_delete ) {
        if ( head != symbol_table )
            temp->se_local_vars = entry_to_delete->se_next;
        else
            symbol_table = entry_to_delete->se_next;
    }
    else {

        /*
         *  Search down the list looking for the element to be deleted.
         */

        while ( head->se_next != entry_to_delete )
            head = head->se_next;

        /*
         *  The next field now points to the element to be deleted, remove the
         *  element from the linked list.
         */

        head->se_next = entry_to_delete->se_next;
    }

    /*
     *  If the entry to be deleted is an Element name (global) and has a local variable
     *  list, free the variables in the list.
     */

    if ( parent == NULL )
        {
        temp = entry_to_delete->se_local_vars;
        while ( temp )
            {
            temp1 = temp;
            temp = temp->se_local_vars;
            free( temp1 );
            }
        }

    /*
     *  Free the space being utilized by the entry that was just deleted.
     */

    free( entry_to_delete );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  decrement_symbol_use_count()
*
*
* MODULE FUNCTION:
*
*  Function decrement_symbol_use_count will decrement the use count of a symbol.  This
*  function exists so that symbols can be deleted.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                                  Release 1.01 - 08/01/91
*                                  Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  decrement_symbol_use_count( parent, name )

    char *parent, *name;

{
    struct symbol_entry *temp;

    /*
     *  Lookup the symbol, if found, decrement the use count.
     */

    if ( temp = lookup_symbol( parent, name ) ) {

        /*
         *  Decrement the use count.  If the value is less than or equal to zero then
         *  the element should be deleted from the symbol table unless the function is
         *  an intrinsic.
         */

        temp->se_use_count--;

        if ((temp->se_use_count > 0) || (temp->se_type & INTRINSIC))
            return( temp->se_use_count );

        /*
         *  Don't delete the entry if it is an Element name (PROCEDURE).  We don't
         *  ever want to delete Elements from the symbol table because their local
         *  variables are in the symbol table.
         */

        if ( temp->se_type & PROCEDURE )
            return( temp->se_use_count );

        /*
         *  We have a symbol table entry whose use count is zero and it is not an
         *  Intrinsic or Element name, so remove the entry.
         */

        remove_symbol_entry( parent, temp );
        return( 0 );
        }
    else
        return( ERR );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  print_symbol_table()
*
*
* MODULE FUNCTION:
*
*   This routine prints the symbol table.  This routine is included for development and
*   debugging purposes.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void print_symbol_table()
{
    struct symbol_entry *temp1, *temp2;

    fprintf( stdout, "\n\nGCB Symbol Table:\n\n" );
    temp1 = symbol_table;

    /*
     *  Print the entries at the root of the symbol table.
     */

    while( temp1 ) {
        fprintf( stdout, "Name: %-15s", temp1->se_symbol );
        fprintf( stdout, "\tType: %-7o", temp1->se_type & SYMBOL_TYPE );
        fprintf( stdout, "\tAttr: %-7o", temp1->se_type & SYMBOL_DATA_TYPE );
        fprintf( stdout, "\t\tUse: %d\n", temp1->se_use_count );

        /*
         *  If the entry has a local variable list (which implies that the entry is
         *  a procedure), print the contents of the local variable list.
         */

        if ( temp1->se_local_vars ) {
            temp2 = temp1->se_local_vars;
            fprintf( stdout, "\n\tLocal Variable List:\n\n" );
            while( temp2 ) {
                fprintf( stdout, "\tName: %-15s", temp2->se_symbol );
                fprintf( stdout, "\tType: %-7o", temp2->se_type & SYMBOL_TYPE );
                fprintf( stdout, "\tAttr: %-7o", temp2->se_type & SYMBOL_DATA_TYPE );
                fprintf( stdout, "\t\t Use: %d\n", temp2->se_use_count );
                temp2 = temp2->se_next;
            }
            fprintf( stdout, "\n" );
        }

        temp1 = temp1->se_next;
    }

    fprintf( stdout, "\nEND of GCB Symbol Table\n\n\n" );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  save_symbol_level()
*
*
* MODULE FUNCTION:
*
*   Function save_symbol_level is invoked to save out a single level of the symbol table
*   to the specified file.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

static int save_symbol_level( symbol_level_head, output_file )

    struct symbol_entry *symbol_level_head;
    FILE *output_file;
{
    int number_entries;
    struct symbol_entry *temp;

    /*
     *  Establish a pointer to the symbol table and count the number of entries on
     *  the current level.
     */

    temp = symbol_level_head;
    number_entries = 0;
    while( temp ) {
        number_entries++;
        temp = temp->se_next;
    }

    /*
     *  Write each entry to the output file, if a procedure is encountered with local
     *  variables then recursively call this routine to write the local variable list.
     */

    if ( fwrite( &number_entries, sizeof( int ), 1, output_file ) == 0 ) {
        fprintf( stderr, "Could not save the number of entries\n" );
        return( -1 );
    }

    temp = symbol_level_head;
    while( temp ) {

        /*
         *  Write out the entry.
         */

        if ( fwrite( temp, sizeof( struct symbol_entry ), 1, output_file ) == 0 ) {
            fprintf( stderr, "Could not write out a symbol entry\n" );
            return( -1 );
        }

        /*
         *  If the entry has local variables, recursively call this routine with a
         *  pointer to the local variable list.
         */

        if ( temp->se_local_vars )
            if ( save_symbol_level( temp->se_local_vars, output_file ) != 0 )
                return( -1 );

        /*
         *  Go to the next entry on the list.
         */

        temp = temp->se_next;
    }

    /*
     *  Issue a successful return code.
     */

    return( 0 );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  symbol_table_save()
*
*
* MODULE FUNCTION:
*
*  Function symbol_table_save will store the symbol table to the specified open file.
*  The file format will be:
*
*     - The number of symbols on the main list will be saved
*     - Each symbol will be saved, if the symbol has a local variable list, a count
*       of the local variables will be saved along with any local variables.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int symbol_table_save( output_file )

    FILE *output_file;
{
    return( save_symbol_level(symbol_table,output_file) );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  restore_symbol_level
*
*
* MODULE FUNCTION:
*
*  Function restore_symbol_level is invoked to restore the symbol table from the
*  specified data file.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

static int restore_symbol_level( input_file, parent )

    FILE *input_file;
    char *parent;
{
    int number_entries, i;
    struct symbol_entry temp, *temp_symbol;

    /*
     *  Read the number of entries on the current level of the symbol table level.
     */

    if ( fread( &number_entries, sizeof( int ), 1, input_file ) == 0 ) {
        fprintf( stderr, "Could not restore the number of entries\n" );
        return( -1 );
    }

    /*
     *  For each entry on the list, read in the data and insert into the symbol
     *  table with the specified parent.  If an item has local variables, recursively
     *  invoke this routine to retrieve the variables.
     */

    for ( i = 0; i < number_entries; i++ ) {

        /*
         *  Read the entry and add to the symbol table list.
         */

        if ( fread( &temp, sizeof( struct symbol_entry ), 1, input_file ) == 0 ) {
            fprintf( stderr, "Could not read a symbol entry\n" );
            return( -1 );
        }

        if ( add_symbol_entry( parent,
                               temp.se_symbol,
                               temp.se_type,
                               temp.se_num_dimensions,
                               temp.se_subs[ SUB1 ],
                               temp.se_subs[ SUB2 ],
                               temp.se_subs[ SUB3 ],
                               temp.se_subs[ SUB4 ] ) != 0 )
            return( -1 );

        temp_symbol = lookup_symbol( parent, temp.se_symbol );
```

```
        temp_symbol->se_use_count = temp.se_use_count;

        /*
         *  If the entry has local variables, recursively call this routine with a
         *  pointer to the local variable list.
         */

        if ( temp.se_local_vars )
            if ( restore_symbol_level( input_file, temp.se_symbol ) != 0 )
                return( -1 );
    }

    /*
     *  Issue a successful return code.
     */

    return( 0 );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  set_sym_attribs()
*
* MODULE FUNCTION:
*
*   This routine is used to the set the INSTALLED attribute of Element file entries
*   in the Comp symbol table.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.01 - 08/01/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  set_sym_attribs( name, attrib, set )

    char    *name;
    int     attrib,
            set;

{
    struct symbol_entry *temp;

    /*
     *  Lookup the symbol, if found, set the attributes.  If the Element filename
     *  is not found in the symbol table, don't worry about it.  The user may be
     *  editing an Element which is in the current Comp directory but is not yet
     *  CALLED or ACTIVATED by the Comp.  In this case, the Element filename would
     *  not yet be in the symbol table.
     */

    if ( temp = lookup_symbol(NULL,name) )
        {
        if ( set )
            temp->se_type |= attrib;
        else
            temp->se_type &= attrib;
        }
    else
        {
        elog(3,"set_sym_attribs: didn't find Element: %s",name);
        return;
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  symbol_table_restore()
*
*
* MODULE FUNCTION:
*
*  Function symbol_table_restore the symbol table from the named file.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int symbol_table_restore( input_file )

    FILE *input_file;
{
    /*
     *  Initialize the current symbol table to NULL and invoke the routine to
     *  recursively recreate the symbol table.
     */

    symbol_table = NULL;
    return( restore_symbol_level( input_file, NULL ) );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  symbol_table_init()
*
*
* MODULE FUNCTION:
*
*  Function symbol_table_init will initialize the symbol table.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  symbol_table_init()
{
    /*
     *  Initialize the symbol table root.
     */

    symbol_table = NULL;

    /*
     *  Add all the intrinsic function names to the root of the symbol table.
     */

    add_symbol_entry( NULL, "cos",  PROCEDURE | INTRINSIC | DOUBLE, (short)0, (short)0,
                                                (short)0, (short)0, (short)0 );
    add_symbol_entry( NULL, "acos", PROCEDURE | INTRINSIC | DOUBLE, (short)0, (short)0,
                                                (short)0, (short)0, (short)0 );
    add_symbol_entry( NULL, "sin",  PROCEDURE | INTRINSIC | DOUBLE, (short)0, (short)0,
                                                (short)0, (short)0, (short)0 );
    add_symbol_entry( NULL, "asin", PROCEDURE | INTRINSIC | DOUBLE, (short)0, (short)0,
                                                (short)0, (short)0, (short)0 );
    add_symbol_entry( NULL, "tan",  PROCEDURE | INTRINSIC | DOUBLE, (short)0, (short)0,
                                                (short)0, (short)0, (short)0 );
    add_symbol_entry( NULL, "atan", PROCEDURE | INTRINSIC | DOUBLE, (short)0, (short)0,
                                                (short)0, (short)0, (short)0 );
    add_symbol_entry( NULL, "power",PROCEDURE | INTRINSIC | DOUBLE, (short)0, (short)0,
                                                (short)0, (short)0, (short)0 );
    add_symbol_entry( NULL, "log",  PROCEDURE | INTRINSIC | DOUBLE, (short)0, (short)0,
                                                (short)0, (short)0, (short)0 );
    add_symbol_entry( NULL, "nlog", PROCEDURE | INTRINSIC | DOUBLE, (short)0, (short)0,
                                                (short)0, (short)0, (short)0 );
    add_symbol_entry( NULL, "exp",  PROCEDURE | INTRINSIC | DOUBLE, (short)0, (short)0,
                                                (short)0, (short)0, (short)0 );
    add_symbol_entry( NULL, "sqrt", PROCEDURE | INTRINSIC | DOUBLE, (short)0, (short)0,
                                                (short)0, (short)0, (short)0 );

    /*
     *  Increment the use count of each intrinsic function.
     */

    increment_symbol_use_count( NULL, "cos");
    increment_symbol_use_count( NULL, "acos");
    increment_symbol_use_count( NULL, "sin");
    increment_symbol_use_count( NULL, "asin");
    increment_symbol_use_count( NULL, "tan");
    increment_symbol_use_count( NULL, "atan");
    increment_symbol_use_count( NULL, "power");
```

```
    increment_symbol_use_count( NULL, "log");
    increment_symbol_use_count( NULL, "nlog");
    increment_symbol_use_count( NULL, "exp");
    increment_symbol_use_count( NULL, "sqrt");
}
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    symbols.c
*
*
* FILE FUNCTION:
*
*   This file contains the routines which create and manipulate the graphical
*   symbols in the work area.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /home/project/3531/Docu/GCB.spec.doc
*
*
* FILE MODULES:
*
*   check_rect()          - checks area for symbols and lines not within area.
*   clear_cell_map_sym()- clears the cells occupied by the parameter area.
*   claim_cells()         - claims the cells to be occupied by a symbol
*   init_cell_map()       - initializes the Cell_Map array.
*   init_symbol_array()   - initializes the Symbol_Map array.
*   install_symbol()      - adds widget with parameter location and type to Symbol_Map
*   next_avail_sym()      - returns first Symbol_Map entry w/o type; sets current_symbol
*   num_cells()           - determines the number of cells a given length will require.
*   remove_symbol()       - deletes symbol from schematic and symbol_map.
*   set_cell_map_sym()    - sets the cells a given symbol will occupy.
*   set_current_sym()     - sets the next available entry to be the parameter widget.
*   set_midpt()           - adjust midpt of symbol canvas to nearest snap point
*   update_pos_fields()   - updates position fields of a Symbol_Map entry after a move.
*   update_symbol_pos()   - moves and redraws a symbol, updates its Symbol_Map fields.
*
*****************************************<----->*****************************************/

#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>

#include "gcb.h"
#include "widgets.h"
#include "lines.h"
#include "element_file.h"
#include "symbol.h"
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  check_rect()
*
* MODULE FUNCTION:
*
*   This routine checks the specified area  for symbols and lines other than
*   lines and symbols within the area or lines entering or leaving the area.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  check_rect( type, cell_x, cell_y, cell_w, cell_h, b_ulcx, b_ulcy, b_w, b_l )

    int     type, cell_x, cell_y, cell_w, cell_h, b_ulcx, b_ulcy, b_w, b_l;

{
    int         y, i, j, x, xx, yy, w, h;
    LineList    *templist;
    Symbol      *from, *to;

    if ( (cell_x < 0) || (cell_y < 0) )
        return( 0 );

    /*
     *  Compute absolute position; make sure it is inside logical work area.
     */

    xx = cell_x * CELL_SIZE;
    yy = cell_y * CELL_SIZE;
    w = cell_w * CELL_SIZE;
    h = cell_h * CELL_SIZE;

    if ( ((xx + w) > 1140) || ((yy + h) > 1140) )
        return( 0 );

    /*
     *  Check each cell in rectangle to be occupied; if something is there:
     *  1) if we are copying, fail since there is no way around the obstacle.
     *  2) if it is a symbol cell and it will move or a line cell and it will
     *      move or be deleted, allow it.
     */

    for ( y=cell_y, i=0; i<cell_h; y++, i++ )
        for ( x=cell_x, j=0; j<cell_w; x++, j++ )
            if ( Cell_Map[y][x].cell_type != NONE )
                {

                if ( type == CopyBox )

                    /*
                     *  always fail on copy
                     */

                    return( 0 );

                switch( Cell_Map[y][x].cell_type )
                    {
```

```
            case SYMBOL_CELL:
                if ( !sym_within_box(Cell_Map[y][x].cell_entry.symbol,
                    b_ulcx, b_ulcy, b_w, b_l) )
                        {
                        /*DEBUG*/
                        elog(3,"sym %i in way, not in box\n",
                        compute_label_index(Cell_Map[y][x].cell_entry.symbol));
                        return( 0 );
                        }
                break;
            case LINE_CELL:
                templist = Cell_Map[y][x].cell_entry.lines;
                while ( templist )
                    {
                    if ( !line_enters_box(templist->line,  b_ulcx, b_ulcy,
                        b_w, b_l) )
                            {
                            from = (Symbol *)templist->line->from;
                            to = (Symbol *)templist->line->to;
                            /*DEBUG*/
                            elog(3,"line connecting %iand%i not in box\n",
                            compute_label_index(from),compute_label_index(to));
                            return( 0 );
                            }
                    templist = (LineList *) templist->next;
                    }
                break;
            }
        }
    return( 1 );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  clear_cell_map_sym()
*
*
* MODULE FUNCTION:  This module clears the cells occupied by the parameter area.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  clear_cell_map_sym( type, cell_x, cell_y, width, height )

    int type,
        cell_x,
        cell_y,
        height,
        width;

{
    register int    i,j,
                    x,y;

    /*
     *  Set cell map to symbol type and set pointer to either symbol structure
     *  or LineList structure.
     */

    for ( y=cell_y, i=0; i<height; y++, i++ )
        for ( x=cell_x, j=0; j<width; x++, j++ )
            {
            Cell_Map[y][x].cell_type = NONE;
            Cell_Map[y][x].cell_entry.symbol = NULL;
            }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  claim_cells()
*
*
* MODULE FUNCTION:
*
*   This routine clears the cells presently occupied by a symbol and
*   claims the cells to be occupied by the moved symbol from the ulcx and y
*   parameters to the limits of the width and height parameters.
*   If the cells can't be claimed, the routine reclaims the symbol's
*   previous position. If it can't do that, it panics and dies.
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int claim_cells( w )

    Widget  w;

{
    Symbol              *symbol;
    XWindowAttributes   attribs;
    int                 cell_width, cell_height, cell_x, cell_y;

    /*
     *  get window x and y
     */

    if (! XGetWindowAttributes(display,XtWindow(w),&attribs) )
        error_handler( ERR_SYM_ATTRIBS, "claim_cells" );

    cell_width  = num_cells( attribs.width ) ;
    cell_height = num_cells( attribs.height );
    cell_x = attribs.x / CELL_SIZE;
    cell_y = attribs.y / CELL_SIZE;

    if (! (symbol = (Symbol *)get_sym_map_entry(w)) )
        {
        user_ack("Fatal Error: can not find sym map entry in claim cells");
        exit( ERR );
        }

elog(3,"claim cells:  clearing cells at %d %d, width %d  height %d\n",
    symbol->cell_x, symbol->cell_y, symbol->cell_width, symbol->cell_height);

    /*
     *  clear current position.
     */

    clear_cell_map_sym( SYMBOL_CELL,
        symbol->cell_x,
        symbol->cell_y,
        symbol->cell_width ,
        symbol->cell_height );

    /*
     *  set new position.
     */

    if (! (set_cell_map_sym( SYMBOL_CELL, (Symbol *)symbol,
        cell_x,     cell_y,
        cell_width , cell_height)) )
        {

        /*
         *   claim failed, back out
         *   Restore previous cells of this symbol
         */

        if (! (set_cell_map_sym( SYMBOL_CELL, (Symbol *) symbol,
                symbol->cell_x,     symbol->cell_y,
                symbol->cell_width , symbol->cell_height)) )
            {
            elog(1,"claim cells: claim failed and can't reclaim old position");
            exit( ERR );
            }
        return( 0 );
        }
    else
        {

        /*
         *  update Symbol_Map fields for this symbol to reflect its new location
         */

        update_pos_fields( symbol, w, attribs.x, attribs.y );
        return( 1 );
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  free_cellmap_symbols()
*
*
* MODULE FUNCTION:
*
*   This routine frees the cells occupied by the parameter symbol.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  free_cellmap_symbols()
{
    register int i,j;

    for ( i=0; i<CELL_ROWS; i++ )
        for (j=0; j<CELL_COLS; j++ )
            if ( Cell_Map[i][j].cell_type == SYMBOL_CELL )
                {
                Cell_Map[i][j].cell_type = NONE;
                Cell_Map[i][j].cell_entry.symbol = NULL;
                }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  init_cell_map()
*
*
* MODULE FUNCTION:
*
*   This routine initializes the Cell_Map array.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  init_cell_map()
{
    register int i,j;

    for ( i=0; i<CELL_ROWS; i++ )
        for ( j=0; j<CELL_COLS; j++ )
            {
            Cell_Map[i][j].cell_type = NONE;
            Cell_Map[i][j].cell_entry.symbol = NULL;
            }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  init_symbol_array()
*
*
* MODULE FUNCTION:
*
*   This routine initializes the Symbol_Map array.
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  init_symbol_array()
{
    register int i;

    for ( i=0; i<MAX_SYMBOLS; i++ )
        {
        Symbol_Map[i].symbol_type    = NONE;
        Symbol_Map[i].next           = NULL;
        }
    Begin_Sym = NULL;
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  install_symbol()
*
*
* MODULE FUNCTION:
*
*   This routine adds the parameter widget with the parameter location and type
*   to the Symbol_Map array.
*   This routines assumes that the X and Y coordinates that are passed to this
*   routine are the upper left coordinates of a cell.  The placement and movement
*   routines must "snap" the symbols into cells during movement.
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  install_symbol( type, x, y, height, width, text, canvas, font )

    int         type,
                x,
                y,
                height,
                width;
    char        *text;
    Widget      canvas;
    Font        font;

{
    int                 cell_height,
                        cell_width,
                        cell_x,
                        cell_y;
    int                 pos,
                        attributes = 0;
    XWindowAttributes   attribs;
    Arg                 args[1];


    /*
     *  Snap ulcx and y to closest snap point
     *  and reset parameters x and y to reflect snapped position
     */

    if ( Snap )
        set_midpt( canvas );

    if (! XGetWindowAttributes(display,XtWindow(canvas),&attribs) )
        error_handler( ERR_SYM_ATTRIBS, "install_symbol" );

    x = attribs.x;
    y = attribs.y;

    /*
     *  Determine cell map x and y.
     */

    cell_x = x / CELL_SIZE;
    cell_y = y / CELL_SIZE;
```

```
    /*
     *  Determine cell width and height.
     */

    cell_width  = num_cells( width );
    cell_height = num_cells( height );

    /*
     *  Grab the cell map cells for this symbol.
     */

    if (!(set_cell_map_sym( SYMBOL_CELL, (Symbol *) &(Symbol_Map[nextsymbol]),
                        cell_x,     cell_y,
                        cell_width , cell_height)))
        {
        if (type == TEXT)
            user_ack("Cells are already occupied, can not place TEXT there");
        else
            user_ack("Cells are already occupied, can not place Symbol there");
        return( ERR );
        }

    Symbol_Map[nextsymbol].cell_x           = cell_x;
    Symbol_Map[nextsymbol].cell_y           = cell_y;
    Symbol_Map[nextsymbol].cell_width       = cell_width;
    Symbol_Map[nextsymbol].cell_height      = cell_height;

    /*
     *  Init line draw pointers.
     */

    if ( type == IF )
        {
        Symbol_Map[nextsymbol].Sym.IfSym.true_line  = NULL;
        Symbol_Map[nextsymbol].Sym.IfSym.false_line = NULL;
        }
    else
        Symbol_Map[nextsymbol].next = NULL;

    Symbol_Map[nextsymbol].from = NULL;

    /*
     *  Set Symbol_Map fields
     */

    Symbol_Map[nextsymbol].symbol_type  = type;
    Symbol_Map[nextsymbol].key          = SymKey++;
    Symbol_Map[nextsymbol].ulcx         = x;
    Symbol_Map[nextsymbol].ulcy         = y;
    Symbol_Map[nextsymbol].height       = height;
    Symbol_Map[nextsymbol].width        = width;
    Symbol_Map[nextsymbol].text         = text;
    Symbol_Map[nextsymbol].mycanvas     = canvas;
    Symbol_Map[nextsymbol].font         = font;

    /*
     *  put text in Symbol_Map entry according to symbol type
     */

    switch( type )
        {

        case IF:
        case SET:
            elog(3,"Ltext: %s\n", Symbol_Map[nextsymbol].Sym.IfSym.logical_expr =
                (char *)sym_text);

            elog(3,"comp: %s\n", Symbol_Map[nextsymbol].Sym.IfSym.comp_expr =
                (char *)expr_text);

            elog(3,"comment: %s\n", Symbol_Map[nextsymbol].Sym.IfSym.comment =
                (char *)comment_text);

            expr_text       = NULL;
            sym_text        = NULL;
            comment_text    = NULL;

            /*
             *  increment use count for each var in this expr
             */

            change_list_use_count( current_symbol,
                Symbol_Map[nextsymbol].Sym.IfSym.comp_expr, 1 );

            break;

        case BEGIN:

            /*
             *  Set BeginSym so code generator will know where to begin.
             */

            Begin_Sym = &Symbol_Map[nextsymbol];
            break;

        case START:
        case STOP:

            if (! (struct symbol_entry *) lookup_symbol(NULL,text) )
                {
                attributes |= COMP_REF;
                if ( add_symbol_entry(NULL,text,attributes,0,0,0,0,0) )
                    error_handler( ERR_ADD_SYMBOL, "install_symbol" );
                }

            /*
             *  Increment reference count for the Comp name.
             */

            increment_symbol_use_count( NULL, text );
            break;

        case GOTO:

            /*
             *  save the comp type in case of delete so we know what type of element
             *  (ELEMENT or LIB) we are restoring.
             */

            Symbol_Map[nextsymbol].Sym.ElemSym.comp_type = CallType;
            elog( 3,"install: call type = %d\n", CallType );

            /*
             *  if not found in sym table, install this element reference
             */

            if (! (struct symbol_entry *) lookup_symbol(NULL,text) )
                {
```

```
            elog(3,"install: element name %s not found; adding to sym tab",text);
            attributes |= PROCEDURE;
            if ( add_symbol_entry(NULL,text,attributes,0,0,0,0,0) )
                error_handler( ERR_ADD_SYMBOL, "install_symbol" );
            }

        /*
         *  increment reference count
         */

        elog(3,"install: element name found; incr use count to %i\n",
        increment_symbol_use_count( NULL, text ) );

        break;
    default:
        break;
    }

if ( type != TEXT )

    /*
     *  element is incomplete.
     */

     upd_status(0);

SaveNeeded = 1;

return( nextsymbol );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  next_avail_sym()
*
*
* MODULE FUNCTION: looks for first entry in Symbol_Map that has no
*          symbol type; sets current_symbol to that entry and returns
*          its number in the Symbol Map.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  next_avail_sym()
{
    int i;

    /*
     *  Check next unused slot out from array of widgets.
     */

    for ( i = 0; i < MAX_SYMBOLS - 1; i++ )
        if ( Symbol_Map[i].symbol_type == NONE )
            {
            nextsymbol = i;
            current_symbol = symbols[nextsymbol] = (Widget)create_new_sym();
            return( nextsymbol );
            }

    /*
     *  couldn't find an available symbol
     */

    user_ack("You have run out of symbols; delete some or increase MAX_SYMBOLS");
    return( ERR );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  num_cells()
*
*
* MODULE FUNCTION:
*
*   This module determines the number of cells a given length will require.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  num_cells( length )

    int length;

{
    if ( length < CELL_SIZE )
        return( 1 );

    if ((length % CELL_SIZE) == 0 )
        return( length / CELL_SIZE );

    return( (length/CELL_SIZE) + 1 );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  remove_symbol()
*
*
* MODULE FUNCTION:
*
*   This routines assumes that the X and Y coordinates that are passed to this
*   routine are the upper left coordinates of a cell.  The placement and movement
*   routines must "snap" the symbols into cells during movement.
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  remove_symbol( symbol )

    Widget symbol;
{
    int                 cell_height,
                        cell_width,
                        cell_x,
                        cell_y,
                        sym_type,
                        sym;
    XWindowAttributes   attribs;
    Arg                 args[1];

    if (! XGetWindowAttributes(display,XtWindow(symbol),&attribs) )
        error_handler( ERR_SYM_ATTRIBS, "remove_symbol" );

    /*
     *  clear symbol map entry
     */

    for ( sym=0; sym<MAX_SYMBOLS; sym++ )
        if ( Symbol_Map[sym].mycanvas == symbol )
            {
elog(3,"removing sym# %d\n", sym);

            /*
             *  set event for undo
             *  set global text variables so draw_symbol has them;
             *  this is necessary since in Motif there is no way to link >1
             *  piece of data with a widget, and the symbol_type is already linked.
             */

            XtSetArg( args[0], XmNuserData, &sym_type );
            XtGetValues( symbol, args, 1 );

            if ( (sym_type == IF) || (sym_type == SET) )
                {
                sym_text = Symbol_Map[sym].Sym.IfSym.logical_expr;
                expr_text = Symbol_Map[sym].Sym.IfSym.comp_expr;
                }
            else
                sym_text = Symbol_Map[sym].text;

            set_symbol_event( Symbol_Map[sym] );
```

```
        /*
         *  delete all lines entering and leaving symbol
         */

        delete_lines( symbol );

        if ( Symbol_Map[sym].symbol_type == BEGIN )
            Begin_Sym = NULL;

        /*
         *  for goto, start and stop: decrement the element/comp use count
         */

        if ( (Symbol_Map[sym].symbol_type == GOTO)  ||
             (Symbol_Map[sym].symbol_type == START) ||
             (Symbol_Map[sym].symbol_type == STOP) )
                {
                elog(3,"remove: decr use count to %i\n",
                decrement_symbol_use_count( NULL, Symbol_Map[sym].text ) );
                }
        else if ( (Symbol_Map[sym].symbol_type == IF) ||
                  (Symbol_Map[sym].symbol_type == SET) )

        /*
         *  delete from the symbol table all references to the variables in
         *  this symbol's expression.
         */

                {

                /*
                 *  set last variable list so undo can restore vars to
                 *  symbol table.
                 */

                set_last_var_list( Symbol_Map[sym].Sym.IfSym.comp_expr );

                change_list_use_count( current_symbol,
                    Symbol_Map[sym].Sym.IfSym.comp_expr, 0 );
                }

        Symbol_Map[sym].symbol_type = NONE;

        XClearWindow( display, XtWindow(symbol) );
        XtUnmapWidget( symbol );

        /*
         *  removing this symbol may change element status to complete
         */

        if ( complete(0, LINES_AND_EXPR) == ERR )
            upd_status( 1 );
        else upd_status( 0 );

        SaveNeeded = TRUE;

        break;
        }

/*
 *  Determine cell map x and y.
 */

cell_x = attribs.x / CELL_SIZE;
cell_y = attribs.y / CELL_SIZE;

/*
 *  Determine cell width and height.
 */

cell_width  = num_cells( attribs.width );
cell_height = num_cells( attribs.height );

/*
 *  clear cells formerly occupied by this symbol.
 */

clear_cell_map_sym( SYMBOL_CELL, cell_x, cell_y, cell_width , cell_height );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  set_cell_map_sym()
*
*
* MODULE FUNCTION:  This module checks the area to be occupied by the symbol;
*       if it is ok, the cells occupied by the symbol are set.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  set_cell_map_sym( type, sym_ptr, cell_x, cell_y, width, height )

    int     type,
            cell_x,
            cell_y,
            height,
            width;

    void    *sym_ptr;

{
    register int    i,j,
                    x,y, z;
    LineList        *llist;
    int             xx, yy, w, h, rejected = 0;
    Symbol          *symbol_ptr;

    /*
     *  can't set cells outside of canvas
     */

    if ( (cell_x < 0) || (cell_y < 0) )
        return( 0 );

    /*
     *  if this claim will fail when we zoom out or back in,
     *  can't be allowed
     */

    xx = cell_x * CELL_SIZE;
    yy = cell_y * CELL_SIZE;
    w = width * CELL_SIZE;
    h = height * CELL_SIZE;

     if ( Zoomed )
        {
        if ( ((xx + w) >= op(1140, 2)) || ((yy + h) >= op(1140,2)) )
            return( 0 );
        }
    else if ( ((xx + w) > 1140) || ((yy + h) > 1140) )
        return( 0 );

    /*
     *  Go through cells first to see if all can be set.
     *  If not, return without changing any.
     */

    for ( y=cell_y, i=0; i<height; y++, i++ )
        for ( x=cell_x, j=0; j<width; x++, j++ )
            {
            if ( (y >= CELL_ROWS) || (x >= CELL_COLS) )
                {
                /*DEBUG*/
                elog(3,"set cell map sym: y or x > cell_rows/cols\n");
                return( 0 );
                }
            else if ( Cell_Map[y][x].cell_type != -1 )
                {
                if ( Mode != GhostMoveBox )
                    {
                    if ( Cell_Map[y][x].cell_type == LINE_CELL )
                        {
                        /*
                         *  allow IF,SET, and PRINT to grow over their own lines
                         *  in edit mode. Everything else is disallowed.
                         */

                        llist = (LineList *)Cell_Map[y][x].cell_entry.lines;
                        symbol_ptr = (Symbol *)sym_ptr;
                        if ( (symbol_ptr->symbol_type != IF) &&
                             (symbol_ptr->symbol_type != SET) &&
                             (symbol_ptr->symbol_type != PRINT) )
                            rejected = 1;


                        /*
                         *  if, set, and print can grow only over their own lines
                         */

                        if ( (llist->line->from != (struct Symbol *) symbol_ptr) &&
                             (llist->line->to   != (struct Symbol *) symbol_ptr) )
                            rejected = 1;

                        /*
                         *  must be editing to allow growth.
                         */

                        if  ( Mode != EditSymbolContents )
                            rejected = 1;

                        if ( rejected )
                            return( 0 );
                        }
                    else
                        return( 0 );
                    }
                }

    /*
     *  Set cell map to symbol type and set pointer to either symbol structure
     *  or LineList structure.
     */

    for ( y=cell_y, i=0; i<height; y++, i++ )
        for ( x=cell_x, j=0; j<width; x++, j++ )
            {
            Cell_Map[y][x].cell_type = type;
            Cell_Map[y][x].cell_entry.symbol = (Symbol *) sym_ptr;
            }
    return( 1 );
```

```
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  set_midpt()
*
*
* MODULE FUNCTION: adjust midpt of symbol canvas to nearest snap point
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  set_midpt( canvas )

    Widget  canvas;

{

    XWindowAttributes   attribs;
    int                 x, y;
    int                 snap_cell = CELL_SIZE * (Zoomed ? 10 : 15);
    int                 midx, midy, snapx, snapy;
    Window              w = XtWindow( canvas );

    if (! XGetWindowAttributes(display,w,&attribs) )
        error_handler( ERR_SYM_ATTRIBS, "set_midpt" );

    x = attribs.x;
    y = attribs.y;

    if ( x < 0 )

        /*
         *  going off the left edge - move symbol to the left edge.
         */

        XMoveWindow( display, w, 0, attribs.y );


    else if ( (x + attribs.width) > (MAIN_CANVAS_WIDTH + MAIN_CANVAS_WIDTH/2 - 350) )

        /*
         *  going off the right edge - move symbol to the right edge.
         */

        XMoveWindow( display, w,
            x - ((x+attribs.width) - (MAIN_CANVAS_WIDTH+MAIN_CANVAS_WIDTH/2 - 350)),
            attribs.y );

    if ( y < 0 )

        /*
         *  going off the top edge
         */

        XMoveWindow( display, w, attribs.x, 0 );


    else if ( (y + attribs.height) > MAIN_CANVAS_HEIGHT + MAIN_CANVAS_HEIGHT/2 )
```

```
        /*
         *  going off the bottom edge
         */

        XMoveWindow( display, w, attribs.x,
            y - ((y + attribs.height)-MAIN_CANVAS_HEIGHT) );


    midx = attribs.x + ( attribs.width / 2 );
    midy = attribs.y + ( attribs.height / 2 );

    /*
     *  compute # of snap_cells in x direction
     */

    snapx = midx / snap_cell;

    /*
     *  compute # of snap_cells in y direction
     */

    snapy = midy / snap_cell;

    /*
     *  reset midpt of symbol to nearest snap pt;
     *  do we go forward or backward in x direction
     */

    if ( !(snapx) ) midx = snap_cell;
    else if ( (midx % snap_cell) <= (snap_cell / 2) )
        midx = snap_cell * snapx;
    else
        midx = snap_cell * snapx + snap_cell;

    /*
     *  do we go up or down in y direction
     */

    if ( !(snapy) ) midy = snap_cell;
    else if ( (midy % snap_cell) <= (snap_cell / 2) )
        midy = snap_cell * snapy;
    else
        midy = snap_cell * snapy + snap_cell;

    /*
     *  set ulcx and y of canvas from new midpt
     */

    XMoveWindow( display, w, midx-(attribs.width/2), midy-(attribs.height/2) );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME: set_current_sym()
*
*
* MODULE FUNCTION: as the name implies, sets the next available entry to be the
*   parameter canvas.
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*
*****************************************<----->*****************************************/

void  set_current_sym( symbol )

    Widget  symbol;
{
    Symbol  *tempsym;

    /*
     *  Set the global var current_symbol to be the parameter widget. This
     *  makes edit, for example, much easier by allowing all functions to work
     *  off the global variable.
     */

    if ( (tempsym = (Symbol *) get_sym_map_entry(symbol)) != NULL )
        current_symbol = tempsym->mycanvas;
    else
        {
        user_ack("Fatal Error: can not set current symbol");
        exit( ERR );
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  update_pos_fields()
*
*
* MODULE FUNCTION:
*
*   This module updates the position fields of a Symbol_Map entry after a move.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  update_pos_fields( symbol, canvas, ulcx, ulcy )

    Symbol      *symbol;
    Widget      canvas;
    int         ulcx, ulcy;

{
    XWindowAttributes   attribs;

    if (! XGetWindowAttributes(display,XtWindow(canvas),&attribs) )
        error_handler( ERR_SYM_ATTRIBS, "update_pos_fields" );

    /*
     *   update position fields in Symbol_Map entry for this symbol
     */

    symbol->ulcx = ulcx;
    symbol->ulcy = ulcy;

    symbol->cell_x = ulcx / CELL_SIZE;
    symbol->cell_y = ulcy / CELL_SIZE;

    symbol->width = attribs.width;
    symbol->height = attribs.height;

    /*
     *  update height and width in case edit has changed them.
     */

    symbol->cell_width = num_cells( attribs.width );
    symbol->cell_height = num_cells( attribs.height );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  update_symbol_pos()
*
*
* MODULE FUNCTION:
*
*   This routines assumes that the X and Y coordinates that are passed to this
*   routine are the upper left coordinates of a cell.  The placement and movement
*   routines must "snap" the symbols into cells during movement.
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  update_symbol_pos( w )

    Widget  w;

{
    Arg                 args[1];
    int                 newx, newy;
    int                 sym_type, sym, found = 0;
    XWindowAttributes   attribs;

    /*
     *  find Symbol_Map entry corresponding to this widget.
     */

    for ( sym=0; sym<MAX_SYMBOLS; sym++ )
        if ( Symbol_Map[sym].mycanvas == w )
            {
            found++;
            break;
            }

    if ( (!(found)) || (Symbol_Map[sym].symbol_type == NONE) )
        {
        elog(3,"update sym pos: couldn't find sym\n");
        return( ERR );
        }
    else
        elog(3,"update sym pos: sym# %d\n", sym);

    /*
     *  set event for undo.  Also,
     *  set global text variables so draw_symbol has them;
     *  this is necessary since in Motif there is no way to link >1
     *  piece of data with a widget, and the symbol_type is already linked.
     */

    XtSetArg( args[0], XmNuserData, &sym_type );
    XtGetValues( w, args, 1 );

    if ( (sym_type == IF) || (sym_type == SET) )
        {
        sym_text = Symbol_Map[sym].Sym.IfSym.logical_expr;
        expr_text = Symbol_Map[sym].Sym.IfSym.comp_expr;
        }
    else
```

```
        sym_text = Symbol_Map[sym].text;

    /*
     *  save original position for undo.
     */

    set_symbol_event( Symbol_Map[sym] );

    /*
     *  Snap ulcx and y to closest snap point
     *  and reset newx and y to reflect snapped position
     */

    if ( Snap )
        set_midpt( w );

    /*
     *  Grab the cell map cells for this symbol.
     */

    if ( !(claim_cells( w )) )
        {

        /*
         *  Move parentcanvas back to old position
         */

        if ( Symbol_Map[sym].symbol_type == TEXT )
            user_ack("Cells are already occupied, can not move TEXT there");
        else
            user_ack("Cells are already occupied, can not move Symbol there");

        XMoveWindow( display,XtWindow(w),Symbol_Map[sym].ulcx,Symbol_Map[sym].ulcy );
        return( 0 );
        }

    delete_lines( w );

    /*
     *  Init line draw pointers.
     */

    if ( Symbol_Map[sym].symbol_type == IF )
        {
        Symbol_Map[sym].Sym.IfSym.true_line  = NULL;
        Symbol_Map[sym].Sym.IfSym.false_line = NULL;
        }
    else
        Symbol_Map[sym].next = NULL;

    return( 1 );
}
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    term_set.h
*
*
* FILE FUNCTION:
*
*   This file is a bitmap of the sets of allowable next tokens at each stage in
*   the parse. A set bit in an array indicates that the corresponding token, listed
*   in next_inputs.h, is allowable at this point.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /home/project/3531/Docu/GCB.spec.doc
*
*
* FILE MODULES:
*
*   N/A
*
*****************************************<----->*****************************************/

int     term_set[24][NUM_BUTTONS] = {

                                /*11*/            /*20*/            /*30*/

/*0*/    {0,0,0,0,0,1,1,0,0,0,0
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    text.c
*
* FILE FUNCTION:
*   Prints and clears text in the work area.
*
*
* SPECIFICATION DOCUMENTS:
*
*   /home/project/3531/Docu/GCB.spec.doc
*
*
* FILE MODULES:
*
*   clear_text()     -   erases a given string at a given position.
*   set_text()       -   draws a given string, including newlines, at a given position.
*
*****************************************<----->*****************************************/

#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>

#include "gcb.h"
#include "widgets.h"
#include "lines.h"
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  clear_text()
*
* MODULE FUNCTION:
*
*   This routine erases a given string at a given position by changing the function
*   of the graphics context to the background color and then redrawing the text.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  clear_text( canvas_pw, x, y, string )

    Widget  canvas_pw;
    int     x,y;
    char    *string;

{
    /*
     *  Set the X function to the background color and then draw the text, this
     *  clear it by making it the same as the background.
     */

    XSetFunction( display, LDgc, GXcopy );
    XSetForeground( display, LDgc, LDbackground );

    XDrawString( display, XtWindow(canvas_pw), LDgc, x, y, string, strlen(string) );

    /*
     *  Reset back to the default mode.
     */

    XSetFunction( display, LDgc, GXxor );
    XSetForeground( display, LDgc, BlackPixel(display,DefaultScreen(display)) ^
                LDbackground );
}
```

```
/*************************************<----->*************************************
*
* MODULE NAME:  set_text()
*
* MODULE FUNCTION:
*
*   This routine draws a given string, including newlines, at a given position.
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*************************************<----->*************************************/

void  set_text( canvas, gc, x, y, string )

    Widget canvas;
    GC     gc;
    int    x,y;
    char   *string;

{
    char   buf[MAX_TEXT],
           *s = string;
    int    i, k,
           j = 0;

    while ( *s )
        {
        k = 0;

        /*
         *  print 1 line at a time into symbol canvas
         */

        for (i=0; ((*s) && (*s!='\n')); i++)
            {
            buf[i] = *s++;
            k++;
            }
        buf[k] = NULL;

        /*
         *  step over new line
         */

        if ( *s == '\n' )
            *s++;

        XDrawString( display, XtWindow(canvas),
            gc, x, y +  j * (Zoomed ? 8 : 17), buf, strlen(buf) );

        /*
         *  null fill buf again
         */

        for ( i=0; i<k; i++ )
            buf[i] = '\0';
        j++;
        }
}
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    tokens.h
*
* FILE FUNCTION:
*
*   This file contains all the strings which are used to locate the context sensitive
*   help text within the various Graphical Comp Builder popups.  These strings are used
*   much like keywords to locate a section of text in the online documenation.
*
*   The order of these strings is dependent on the #define's in constants.h   The
*   #define's in constants.h are used to index into this character array.  Any changes
*   to this array must also be made to the #define's in constants.h
*
*
* SPECIFICATION DOCUMENTS:
*
*   /home/project/3531/Docu/GCB.spec.doc
*
*
* FILE MODULES:
*
*   N/A
*
*****************************************<----->*****************************************
/


char *tokens[] = {

                    "BEGIN",
                    "END",
                    "IF",
                    "SET",
                    "PAUSE",
                    "CALL",
                    "ACTIVATE",
                    "STOP",
                    "PRINT",
                    "TEXT",
                    "Palette Area",
                    "Work Area",
                    "Welcome",
                    "Ask",
                    "Copy Element",
                    "Create Element",
                    "Select Element",
                    "Print Element",
                    "Target Language",
                    "Delete Element",
                    "Set Colors",
                    "Create Comp",
                    "Select Comp",
                    "Create Position",
                    "Select Position",
                    "Logical Popup",
                    "Call Element",
                    "Pause Popup",
                    "Text Popup",
                    "Defined Functions",
                    "Object Access",
                    "Comp Purpose",
                    "Symbol Attributes",
                    "Variable Selection",
                    "Enter Number",
                    "Status Popup",
                    "Select Root Element",
                    "Displayer",
                    "User Acknowledge",
                    "Work Station Globals",
                    "Enter String",
                    "Activate-Stop Comp",
                };
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    undo.c
*
*
* FILE FUNCTION:
*
*   This file contains the routines which save symbol and line placement events
*   and restore the most recently added or deleted line or symbol
*
*
* SPECIFICATION DOCUMENTS:
*
*   /home/project/3531/Docu/GCB.spec.doc
*
*
* FILE MODULES:
*
*   copy_line               - returns a copy of the line parameter
*   copy_lineseg()          - returns a copy of the lineseg parameter
*   free_linelists()        - free all lines associated with a symbol.
*   free_segments()         - free all segments in a line.
*   restore_line()          - restores the parameter line with its labels, if any
*   restore_moved_symbol()  - restores moved symbol to original site; redraws lines.
*   restore_symbol()        - adds/deletes the last symbol to be deleted/added.
*   restore_to_line()       - redraws the line(s) leaving a symbol.
*   set_line_event()        - save the most recent line event
*   set_symbol_event()      - save the most recent symbol event
*   set_null_undo_event()   - set last event to null so we don't undo when we can't.
*   undo()                  - reverses the most recent line or symbol op
*
*****************************************<----->*****************************************/

#include <stdio.h>
#include <X11/Xlib.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>

#include "gcb.h"
#include "widgets.h"
#include "lines.h"
#include "element_file.h"

#define SYMBOL 1
#define LINE 2
#define MOVE 3

/*
 *  structure definitions
 */

typedef struct
    {
    LineSeg     *line;
    Symbol      to, from;
    } linestruct;       /*line that was added or deleted, w/ its symbols*/

/*
 *  one of the linetypelist structures matches each logical line in the
 *  linelist structure and identifies the type of the line
 */

typedef struct
    {
        int                 type;
        struct linetypelist *next;
    } linetypelist;

/*
 *  this structure records the last event if it was a symbol event
 */

typedef struct
    {
        linetypelist    *type;
        LineList        *ll;
        Line            *next;
        Line            *true;
        Line            *false;
        Symbol          symbol;
    } symbolstruct;     /*symbol that was added or deleted, with its lines*/

/*
 *  this structure records the last event
 */

struct  Event_type
    {
    int                 type;       /*symbol or line*/
    int                 add;        /*true for add, false for delete */
    int                 x_offset;
    int                 y_offset;   /*scrollbar offsets at time of event*/
    union {
        symbolstruct    ss;     /*symbol + lines that was added or deleted*/
        linestruct      ls;     /*line that was added or deleted*/
        } event;
    } last_event;

/*
 *  global variables
 */

int     restoring = 0;
int     moved_symbol = 0;
Symbol  old_symbol;

/*
 *  function prototypes
 */

LineSeg     *copy_lineseg();
Line        *copy_line();
void        restore_line(), free_linelists(), restore_to_line(), restore_symbol();
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  set_line_event()
*
* MODULE FUNCTION:
*
*   This routine records the last event as a line add/delete.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  set_line_event( line )

    Line *line;

{
    Symbol *symbol;

    if ( !((line) && (line->line)) )
        {
        user_ack("invalid line ptr");
        exit( ERR );
        }

    last_event.type = LINE;

    /*
     *  Record whether we are adding or deleting a line.
     */

    if ( (Mode == LineDraw) || (restoring) )
        last_event.add = TRUE;
    else last_event.add = FALSE;

    symbol = (Symbol *)line->from;

    if ( !(symbol) )
        user_ack("no from symbol");

    /*
     *  Record symbols connected by this line.
     */

    last_event.event.ls.from = *(Symbol *)line->from;
    last_event.event.ls.to = *(Symbol *)line->to;

    /*
     *  Copy line; record whether it is a true/false or next line.
     */

    if ( symbol->symbol_type == IF )
        {
        if ( line == symbol->Sym.IfSym.false_line )
            {
            last_event.event.ls.line =
                (LineSeg *)copy_lineseg(symbol->Sym.IfSym.false_line->line);
            last_event.event.ls.from.Sym.IfSym.true_line = NULL;
            }
        else if ( line == symbol->Sym.IfSym.true_line )
            {
            last_event.event.ls.line =
                (LineSeg *)copy_lineseg(symbol->Sym.IfSym.true_line->line);
            last_event.event.ls.from.Sym.IfSym.false_line = NULL;
            }
        else
            {
            elog(1,"set_line_event: line from if sym with no true or false line");
            exit( ERR );
            }
        }
    else last_event.event.ls.line =
        (LineSeg *)copy_lineseg( line->line );
}
```

```
/*************************************<----->*************************************
*
* MODULE NAME:  set_symbol_event()
*
* MODULE FUNCTION:
*
*   This routine records the last event as a symbol add/delete/move.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*************************************<----->*************************************/

void  set_symbol_event( symbol )

    Symbol symbol;

{
    int                 sym;
    LineList            *templist, *newlist;
    linetypelist        *newtype;
    Symbol              *temp;

    elog(3,"setting symbol event");

    last_event.type = SYMBOL;

    /*
     *  Record whether we are adding or deleting a symbol.
     */

    if ( Mode == AddSymbol )
        last_event.add = TRUE;
    else if ( Mode == DragSymbol )
        last_event.add = MOVE;
    else last_event.add = FALSE;
    last_event.event.ss.symbol = symbol;

    /*
     *  find Symbol_map entry corresponding to last_event's canvas
     */

    for ( sym=0; sym<MAX_SYMBOLS; sym++ )
        if ( Symbol_Map[sym].mycanvas == last_event.event.ss.symbol.mycanvas )
            break;

    if ( symbol.symbol_type == IF )
        {

        /*
         *  copy true and false lines
         */

        if ( symbol.Sym.IfSym.true_line )
            last_event.event.ss.true =
                (Line *) copy_line( symbol.Sym.IfSym.true_line);
        else last_event.event.ss.true = NULL;
        if ( symbol.Sym.IfSym.false_line )
            last_event.event.ss.false =
                (Line *) copy_line( symbol.Sym.IfSym.false_line );
        else last_event.event.ss.false = NULL;
        }
    else if ( symbol.next )

        /*
         *  copy next line
         */

        last_event.event.ss.next = (Line *)copy_line( symbol.next );

    else if ( !(symbol.next) )
        last_event.event.ss.next = NULL;

    /*
     *  Copy list of lines entering this symbol
     */

    last_event.event.ss.ll = NULL;
    last_event.event.ss.type = NULL;
    if ( symbol.from )
        {
        templist = symbol.from;
        while ( templist )
            {
            newlist = (LineList *)malloc( sizeof(LineList) );

            newlist->line = (Line *)copy_line( templist->line );
            newlist->key  = templist->key;

            newlist->next = (struct LineList *)last_event.event.ss.ll;
            last_event.event.ss.ll = (LineList *)newlist;

            newtype = (linetypelist *) malloc( sizeof(linetypelist) );

            /*
             *  identify the type of the line being saved and
             *  create a linetypelist entry to record it
             */

            temp = (Symbol *)templist->line->from;
            if ( temp->symbol_type == IF )
                {

                /*
                 *  if the from symbol has a true line
                 *  and it is the line we are copying,
                 *  create a new linetypelist entry reading true
                 */

                if ( (temp->Sym.IfSym.true_line) &&
                    (temp->Sym.IfSym.true_line == templist->line) )
                        newtype->type = TRUE_PTR;

                /*
                 *  ditto for false line
                 */

                else if ( (temp->Sym.IfSym.false_line) &&
                    (temp->Sym.IfSym.false_line == templist->line) )
                        newtype->type = FALSE_PTR;
                else
                    {
                    user_ack("can't find type of line entering this symbol");
                    exit( ERR );
```

```
            }
        }
        else newtype->type = NEXT_PTR;

        newtype->next = (struct linetypelist *)last_event.event.ss.type;
        last_event.event.ss.type = (linetypelist *)newtype;

        templist = (LineList *)templist->next;
        }
    }

/*
 *  maintain moved_symbol - if we are getting new symbol event,
 *  we can't be looking at a moved symbol
 */

moved_symbol = FALSE;
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  set_null_undo_event()
*
* MODULE FUNCTION:
*
*   This routine sets the last_event_type to 0. Sometimes undo should not attempt
*   to undo the previous action; this ensures that it won't.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  set_null_undo_event()

{
    last_event.type = 0;
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  undo()
*
* MODULE FUNCTION:
*
*   This routine undoes the previous add/delete/move of a symbol or a line.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  undo()

{
    Symbol          from, to;
    LineSeg         *lineseg;

    if ( last_event.type == 0 )

        /*
         *  if undo is called before any action is taken, just return
         */

        return;

    if ( last_event.type == SYMBOL )
        {
        elog(3,"last event type is sym");
        restore_symbol();
        }
    else

        /*
         *  lastevent is of LINE type.
         */

        {
        if ( last_event.add )
            {

            /*
             *  record that we are deleting a line.
             */

            restoring = 0;

            /*
             *  set global line pts
             */

            lineseg = last_event.event.ls.line;

            /*
             *  set LDline[XY] to first cell in this line that is of line type
             */

            find_line_cell( lineseg );

            /*
             *  Remember the line we are deleting so we can undo the undo.
             */

            set_line_event( Cell_Map[LDlineY][LDlineX].cell_entry.lines->line );

            delete_line();

            }
        else
            {

            /*
             *  record that we are adding a line.
             */

            restoring = 1;

            /*
             *  restore line and update fields in symbols it connects
             */

            lineseg = last_event.event.ls.line;
            from = last_event.event.ls.from;
            to = last_event.event.ls.to;

            if ( from.symbol_type == IF )
                {
                if ( from.Sym.IfSym.true_line )
                    restore_line( lineseg, 1, from.mycanvas, to.mycanvas );
                else if ( from.Sym.IfSym.false_line )
                    restore_line( lineseg, 2, from.mycanvas, to.mycanvas );
                }
            else restore_line( lineseg, 0, from.mycanvas, to.mycanvas );

            /*
             *  Record this line event for undo
             */

            set_line_event( LDlinePtr );

            /*
             *  if we undo this undo, record that we last added a line.
             */

            restoring = 0;
            }
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  copy_line and copy_lineseg()
*               free_linelists and free_segments()
*
* MODULE FUNCTION:
*
*   As the self_documenting names imply, these routines copy the parameter
*   lineseg or line into a newly allocated structure
*   and free the line and line segment structures on each new setting of
*   last_event.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

LineSeg  *copy_lineseg( lineseg )

    LineSeg *lineseg;
{
    LineSeg *locseg = lineseg;
    LineSeg *new, *temp = NULL;

    /*
     *  get to last segment of line
     */

    if ( locseg )
        while ( locseg->next )
            locseg = (LineSeg *)locseg->next;

    /*
     *  save copy of logical line structure
     */

    while ( locseg )
        {
        new = (LineSeg *) malloc( sizeof(LineSeg) );
        *new = *locseg;
        new->next = (struct LineSeg *)temp;
        temp = (LineSeg *)new;
        locseg = (LineSeg *)locseg->prev;
        }

    return temp;
}


Line *copy_line( line )

    Line *line;
{
    Line *new;

    new = (Line *)malloc( sizeof(Line) );
    new->line = (LineSeg *)copy_lineseg( line->line );
    new->from = line->from;
    new->to   = line->to;
    new->key  = line->key;

    return new;
}


void free_segments( lineseg )

    LineSeg *lineseg;

{
    LineSeg *savPtr;

    while ( lineseg )
            {

            /*
             *   save a pointer to the current line segment
             *   structure in order to free it.
             */

            savPtr = lineseg;
            lineseg = (LineSeg *) lineseg->next;
            free( savPtr );
            }
}


void free_line( line )

    Line *line;
{

    free_segments( line->line );
    free( line );
}


void free_linelists()

{

    LineList *newlist, *savelist;
    Line    *saveline;

    if ( last_event.type == SYMBOL )

        /*
         *  for last_event's next line list, go thru segs,
         *  free each one, then free line.
         */

        {
        if ( last_event.event.ss.symbol.symbol_type == IF )
            {
            if ( last_event.event.ss.true )
                free_line( last_event.event.ss.true );
            if ( last_event.event.ss.false )
                free_line( last_event.event.ss.false );
            }
        else if ( last_event.event.ss.next )
            free_line( last_event.event.ss.next );

        /*
         *  go thru last_event's 'from' linelist; for each line, go thru segs,
```

```
        *  free each one, then free line.
        *  finally, free linelist.
        */

       newlist = last_event.event.ss.ll;
       while ( newlist )
           {
           saveline = newlist->line;
           free_line( saveline );
           savelist = newlist;
           newlist = (LineList *)newlist->next;
           free( savelist );
           }   /*while*/
       }       /*if*/
   else

       /*
        *  lastevent was a line add or delete; free the malloc'ed
        *  line segment structures
        */

       free_segments(last_event.event.ls.line);
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  restore_line()
*
* MODULE FUNCTION:
*
*   This routine restores the parameter line with its labels, if any, and
*   updates the to and from symbol structures
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  restore_line( lineseg, type, from, to )

    LineSeg     *lineseg;
    int         type;
                    /*
                    * 0 for NEXT
                    * 1 for TRUE
                    * 2 for FALSE
                    */
    Widget      from, to;

{
    /*
     *  Set global line draw variables.
     */

    LDstartX = lineseg->start_x;
    LDstartY = lineseg->start_y;
    LDendX = lineseg->end_x;
    LDendY = lineseg->end_y;

    /*
     *  Set global line draw begin/end symbols.
     */

    if ( (LDendPtr = (Symbol *)get_sym_map_entry(to)) == NULL)
        {
        user_ack("restore line: can't find LDendPtr!");
        exit( ERR );
        }
    if ( (LDstartPtr = (Symbol *)get_sym_map_entry(from)) == NULL)
        {
        user_ack("restore line: can't find LDstartPtr!");
        exit( ERR );
        }

    /*
     *  Go through the same start_line, mid_line, end_line process we go
     *  through when drawing a line through the interface.
     */

    if ( type == 1 )
        start_Line( LDside = TRUE_PTR );
    else if ( type == 2 )
        start_Line( LDside = FALSE_PTR );
    else if ( type == 0 )
```

```
        start_Line( LDside = NEXT_PTR );
    else
        {
        user_ack("invalid line type in restore!");
        exit( ERR );
        }

    while ( lineseg )
        {
        draw_line();
        if ( lineseg = (LineSeg *)lineseg->next )
            {
            mid_Line();

            LDstartX = lineseg->start_x;
            LDstartY = lineseg->start_y;
            LDendX = lineseg->end_x;
            LDendY = lineseg->end_y;
            }
        }
    end_Line();
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  restore_moved_symbol()
*
* MODULE FUNCTION:
*
*   This routine restores a moved symbol to its original location and redraws
*   the lines that were deleted when it was moved.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  restore_moved_symbol( symbol )

    Symbol  *symbol;

{
    int                 sym,i;
    Arg                 args[1];
    XWindowAttributes   attribs;

    XtSetArg( args[0], XmNuserData, &sym );
    XtGetValues( last_event.event.ss.symbol.mycanvas , args, 1 );

    /*
     *  ignore last_event; restore symbol to its
     *  new, moved location with no lines.
     *  delete moved symbol's lines
     */

    delete_lines( last_event.event.ss.symbol.mycanvas );

    XMoveWindow( display, XtWindow(last_event.event.ss.symbol.mycanvas),
                 old_symbol.ulcx,
                 old_symbol.ulcy );

    /*
     *  clear Symbol_Map.symbol_type for reinstallation
     */

    symbol->symbol_type = NONE;

    nextsymbol = compute_label_index( symbol );

    if (! XGetWindowAttributes(display,
        XtWindow(last_event.event.ss.symbol.mycanvas), &attribs) )
            error_handler( ERR_SYM_ATTRIBS, "restore ms" );

    if ( (sym == GOTO) || (sym == START) || (sym == STOP) )
        CallType = last_event.event.ss.symbol.Sym.ElemSym.comp_type;
    if ( (i = install_symbol( sym,
        attribs.x,
        attribs.y,
        attribs.height,
        attribs.width,
        last_event.event.ss.symbol.text,
        old_symbol.mycanvas,
        last_event.event.ss.symbol.font)) < 0 )
```

```
        user_ack("restore symbol failed");

    /*
     *  record that this is a restore of the
     *  move, so to undo this restore, we must
     *  use last_event, since last_event
     *  recorded the lines leading to and from the originial location.
     */

    moved_symbol = FALSE;
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  restore_symbol()
*
* MODULE FUNCTION:
*
*   This routine adds/deletes the last symbol to be deleted/added.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  restore_symbol()

{
    int                 sym,i;
    LineSeg             *lineseg;
    LineList            *templist;
    linetypelist        *temptype;
    Symbol              *symbol, from, to;
    Arg                 args[5];
    XWindowAttributes   attribs;
    extern int          compute_label_index();

    if ( last_event.add == TRUE )
        {

        /*
         *  last event was symbol add; delete symbol
         */

        remove_symbol( last_event.event.ss.symbol.mycanvas );
        }
    else
        {

        /*
         *  retrieve type of last added symbol.
         */

        XtSetArg( args[0], XmNuserData, &sym );
        XtGetValues( last_event.event.ss.symbol.mycanvas , args, 1 );

        symbol=(Symbol *)get_sym_map_entry( last_event.event.ss.symbol.mycanvas);

        if ( last_event.add == MOVE )
            {

            /*
             *  remove symbol from its new location, restore to its previous:
             *  clear cells at present location
             */

            clear_cell_map_sym( SYMBOL_CELL,
                symbol->cell_x,
                symbol->cell_y,
                symbol->cell_width,
                symbol->cell_height );
```

```
        /*
         *  if this symbol is the same as previous restored symbol,
         *  we are restoring a moved symbol - user moved symbol,
         *  undid the move, then undid the undo
         */

        if ( moved_symbol )
            {
            restore_moved_symbol( symbol );
            return;
            }

        else
            {

            /*
             *  restoring original location - use last_event
             */

            XMoveWindow( display,
                XtWindow(last_event.event.ss.symbol.mycanvas),
                last_event.event.ss.symbol.ulcx,
                last_event.event.ss.symbol.ulcy );

            old_symbol = *(Symbol *)symbol;

            /*
             *  clear Symbol_Map.symbol_type for reinstallation
             */

            symbol->symbol_type = NONE;
            }
        }

    /*
     *  set global text fields for the draw_symbol routine
     */

    if ( (sym == IF) || (sym == SET) )
            {
            sym_text = symbol->Sym.IfSym.logical_expr;
            expr_text = symbol->Sym.IfSym.comp_expr;
            }
        else
            sym_text = symbol->text;

    draw_symbol( sym,  last_event.event.ss.symbol.mycanvas, WAgc,
        last_event.event.ss.symbol.font );

    nextsymbol = compute_label_index( symbol );

    if (! XGetWindowAttributes(display,
        XtWindow(last_event.event.ss.symbol.mycanvas), &attribs) )
            error_handler( ERR_SYM_ATTRIBS, "restore sym" );

    if ( (sym == GOTO) || (sym == START) || (sym == STOP) )
        CallType = last_event.event.ss.symbol.Sym.ElemSym.comp_type;

    /*
     *  reinstall this symbol with its attributes.
     */

    if ( (i = install_symbol( sym,
         attribs.x,
         attribs.y,
         attribs.height,
         attribs.width,
         last_event.event.ss.symbol.text,
         last_event.event.ss.symbol.mycanvas,
         last_event.event.ss.symbol.font)) < 0 )
            user_ack("restore symbol failed");

    /*
     *  reinstall the line leaving this symbol
     */

    restore_to_line( sym );

    if ( last_event.event.ss.ll )

        /*
         *  go through recorded line list; restore each recorded line
         *  and update fields in the symbols the line connects.
         */

        {
        templist = last_event.event.ss.ll;
        temptype = last_event.event.ss.type;
        while ( templist )
            {
            from = *(Symbol *)templist->line->from;
            to = *(Symbol *)templist->line->to;
            lineseg = (LineSeg *)templist->line->line;
            if ( temptype->type == TRUE_PTR )
                restore_line( lineseg, 1, from.mycanvas, to.mycanvas );
            else if ( temptype->type == FALSE_PTR )
                restore_line( lineseg, 2, from.mycanvas, to.mycanvas );
            else restore_line( lineseg, 0, from.mycanvas, to.mycanvas );
            templist = (LineList *)templist->next;
            temptype = (linetypelist *)temptype->next;
            }
        }

    /*
     *  set mode to addsymbol so new last_event will be
     *  of add type, unless last_event was a move, in which
     *  case set Mode to Drag so it will be recorded as a move
     */

        if ( last_event.add != MOVE )
            {
            Mode = AddSymbol;

            /*
             *  record this event for undo
             */

            set_symbol_event( Symbol_Map[i] );

            /*
             *  restore previous mode
             */

            Mode = EditSymbol;
            }

    /*
     *  record that this is a re-install of a moved symbol
```

```
        */

            moved_symbol = TRUE;

        /*
         *  show the reinstalled symbol.
         */

            XtMapWidget( last_event.event.ss.symbol.mycanvas );
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  restore_to_line()
*
* MODULE FUNCTION:
*
*   This routine redraws the line(s) leaving a symbol.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  restore_to_line( sym )

    int sym;

{
    Symbol          from, to;
    LineSeg         *lineseg;

    /*
     *  draw line(s) from symbol the way it was recorded,
     */

    if ( sym == IF )
        {

        /*
         *  restore true and false lines, if any.
         */

        if ( last_event.event.ss.true )
            {
            if ( lineseg = (LineSeg *)last_event.event.ss.true->line )
                {
                from = *(Symbol *)last_event.event.ss.true->from;
                to = *(Symbol *)last_event.event.ss.true->to;
                restore_line( lineseg, 1, from.mycanvas, to.mycanvas );
                }
            }
        if ( last_event.event.ss.false )
            {
            if ( lineseg = (LineSeg *)last_event.event.ss.false->line )
                {
                from = *(Symbol *)last_event.event.ss.false->from;
                to = *(Symbol *)last_event.event.ss.false->to;
                restore_line( lineseg, 2, from.mycanvas, to.mycanvas );
                }
            }
        }
    else
        {

        /*
         *  determine symbols this line connects; restore next line.
         */

        if ( last_event.event.ss.next )
            {
```

```
            from = *(Symbol *)last_event.event.ss.next->from;
            to = *(Symbol *)last_event.event.ss.next->to;
            lineseg = (LineSeg *)last_event.event.ss.next->line;
            restore_line( lineseg, 0, from.mycanvas, to.mycanvas );
            }
        }
}
```

```
/*****************************************<----->*****************************************
*
* FILE NAME:    utils.c
*
* FILE FUNCTION:
*
*   This file contains the routines which augment the routines in the other files.
*
*
* FILE MODULES:
*
*   arm_tgl()            - set the parameter toggle active.
*   ask()                - gets yes/no response from user.
*   busy()               - changes cursor to clock.
*   complete()           - determines if the element is complete
*   compute_label_index()  - determines index of symbol into Symbol_Map array.
*   disarm_tgl()         - set the parameter toggle inactive.
*   display_file()       - fills the help window with text from the help file.
*   elog()               - writes a message to the error log file
*   error_handler()      - called when the GCB detects errors.
*   get_filename_ptr()   - extracts from a path name the name before the last /
*   get_sym_map_entry()  - given a widget, returns its Symbol_Map entry.
*   load_font()          - performs the XLib calls to load the specified font.
*   load_help_file()     - searches the help file for the parameter string.
*   open_read_defaults()- reads from user's defaults file.
*   process_popup()      - manages user_ack; does local processing til user responds
*   redraw_sym_num()     - redraws symbol with specified index into Symbol_Map.
*   redraw_sym_type()    - redraws all symbols of specified type.
*   set_attribs()        - sets width, height, and resize policy resources
*   set_position()       - sets position of widget using Position resource.
*   set_user_data()      - sets XmNUserData resource of parameter widget.
*   upd_mode_panel()     - updates mode panel
*   upd_pos_panel()      - updates uppper left area with user name, etc.
*   upd_status()         - updates status area
*   user_ack()           - pops up a message requiring user response.
*   whirl()              - draws | or - to show GCB's progress in initialization.
*   write_defaults()     - at exit, writes current settings to defaults file.
*
*****************************************<----->*****************************************/

#include <stdio.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include "gcb.h"
#include "widgets.h"
#include "constants.h"
#include "element_file.h"
#include "symbol.h"
#include "gcb_parse.h"
#include "cursors.h"

extern int  PopupStat[];

FILE  *openFile( );
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  arm_tgl()
*
* MODULE FUNCTION:
*
*   This routine set the parameter toggle active.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  arm_tgl(tgl)

    Widget tgl;

{
    XmToggleButtonSetState( tgl, True, False );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  ask()
*
* MODULE FUNCTION:
*
*   This routine gets yes/no response from user.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  ask( prompt )

    char    *prompt;

{
    Arg     args[1];
    extern  int Ask_Resp;
    void    process_popup();

    /*
     *  Setup the prompt string.
     */

   XtSetArg( args[0], XmNlabelString,
        XmStringCreate(prompt,XmSTRING_DEFAULT_CHARSET) );
    XtSetValues( lbl_ask, args, 1 );

    /*
     *  Set the size (width) of the prompt.
     */

        XtSetArg( args[0], XmNwidth, (200+(8*strlen(prompt))) );

    XtSetValues( frm_ask, args, 1 );

    process_popup( dlg_ask, WAIT );
    return( Ask_Resp );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  busy()
*
* MODULE FUNCTION:
*
*   This routine changes cursor to clock.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  busy( win, flag )

    Widget      win;
    int         flag;

{
    if ( flag )
        XDefineCursor( display, XtWindow(win), clock_cursor );
    else
        XUndefineCursor( display, XtWindow(win) );

    XFlush( display );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  complete()
*
* MODULE FUNCTION:
*
*   This routine determines if the element is complete; if it has a begin and at
*   least 1 end, and all its expressions are complete and correct.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int complete(start, type)

    int start, type;

{
    int i, end_present = FALSE, j;

    /*
     *  if there is a BeginSym and at least 1 End and every symbol
     *  except the Ends has a next line and every symbol except the Begin has a line
     *  entering it, and the expressions in the IF/SETs are complete, comp status is
     *  complete.
     */

    if ( type == JUST_EXPR )
        {

        /*
         *  if we are just checking expressions, make sure that each
         *  expression is syntactically correct and complete.
         */

        for ( i=start; i<MAX_SYMBOLS; i++ )
            if ( Symbol_Map[i].symbol_type != NONE )
                if ( (Symbol_Map[i].symbol_type == IF) ||
                    (Symbol_Map[i].symbol_type == SET) )
                    {

                    /*
                     *  set global var so parser will know what kind of
                     *  expression to expect.
                     */

                    if ( Symbol_Map[i].symbol_type == IF )
                        SetSym = 0;
                    else
                        SetSym = 1;
                    if ( (parse_expression(Symbol_Map[i].Sym.IfSym.comp_expr)) !=
                        PARSE_SUCCESS )
                            return( i );
                    }
        return( ERR );
        }

    if ( !(Begin_Sym) )
        return( NO_BEGIN );

    /*
     *  find >= 1 end symbol
     */

    for ( i=0; i<MAX_SYMBOLS; i++ )
        if ( Symbol_Map[i].symbol_type == END )
            {
            end_present = TRUE;
            break;
            }

    if ( !(end_present ))
        return( NO_END );

    for ( i=start; i<MAX_SYMBOLS; i++ )
        if ( Symbol_Map[i].symbol_type != NONE )
            {

            /*
             *  line entering this symbol?
             */

            if ( (Symbol_Map[i].from == NULL) &&
                ( (Symbol_Map[i].symbol_type != BEGIN) &&
                    (Symbol_Map[i].symbol_type != TEXT) ) )
                        return( i );
            else if ( Symbol_Map[i].symbol_type == IF )
                {

                /*
                 *  line leaving this IF symbol?
                 */

                if ( (Symbol_Map[i].Sym.IfSym.true_line == NULL) ||
                    (Symbol_Map[i].Sym.IfSym.false_line == NULL) )
                        return( i );
                }
            else if ( (Symbol_Map[i].next == NULL) &&
                    ( (Symbol_Map[i].symbol_type != END) &&
                        (Symbol_Map[i].symbol_type != TEXT) ) )
                        {

                    /*
                     *  no line leaving this symbol.
                     */

                        return( i );
                        }

            if ( type == LINES_AND_EXPR )
                {

                /*
                 *  check for correctness and completeness of expression.
                 */

                if ((Symbol_Map[i].symbol_type==IF) || (Symbol_Map[i].symbol_type==SET))
                    {

                    /*
                     *  set global var so parser will know what kind of
                     *  expression to expect.
                     */
```

```
                if ( Symbol_Map[i].symbol_type == IF )
                    SetSym = 0;
                else
                    SetSym = 1;
                j = parse_expression( Symbol_Map[i].Sym.IfSym.comp_expr );
                if ( j != PARSE_SUCCESS )

                    /*
                     *  this symbol's expression is incomplete.
                     */

                    return( i );
                }
            }
        }

    return( ERR );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  compute_label_index()
*
* MODULE FUNCTION:
*
*   Function compute_label_index will determine the index of the symbol into the
*   global array.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  compute_label_index( symbol )

    Symbol *symbol;

{
    /*
     *  Compute the offset of the parameter into the Symbol_Map array.
     */

    return( ( (int)symbol - (int)Symbol_Map ) / sizeof( Symbol ) );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  disarm_tgl()
*
* MODULE FUNCTION:
*
*   This routine set the parameter toggle inactive.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  disarm_tgl(tgl)

    Widget tgl;

{
    XmToggleButtonSetState( tgl, False, False );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  display_file()
*
* MODULE FUNCTION:
*
*   This routine fills the help window with text from the help file.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  display_file(mode, file)

    int     mode;
    char    *file;

{
     FILE           *fp;

    register int    i   = 0,
                    ptr = 0;

    char            c,
                    string[101];

    Arg             args[1];

    /*
     *  Open the file.
     */

    if ( (fp = (FILE *)openFile( file, "r" )) == NULL )
        {
        elog(1,"display_file: open file failed");
        return;
        }

    /*
     *   set help text widget editable.
     */

    XtSetArg( args[0], XmNeditable, TRUE );
    XtSetValues( txt_file, args, 1 );

    /*
     *  Read data from the file. Read 100 bytes at a time and add to the text widget's
     *  string.
     */

    XmTextSetString( txt_file, NULLS );
    while ( ptr != EOF )
        {
        while ( i < 100 && ( string[ i ] = c = getc ( fp ) ) != EOF )
            i++;
        string[ i ] = NULL;
        XmTextReplace ( txt_file, ptr, ptr, string );
        if ( c == EOF )
            ptr = EOF;
```

```
        else
            {
            ptr += i;
            i = 0;
            }
        }

    /*
     *  Close the file and setup the help string.
     */

    fclose( fp );

    /*
     *   set help text widget ineditable.
     */

    XtSetArg( args[0], XmNeditable, FALSE );
    XtSetValues( txt_file, args, 1 );

    XtManageChild( dlg_file );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  elog()
*
* MODULE FUNCTION:
*
*   This routine prints a formatted string to the user's log file.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.01 - 08/01/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  elog( level, format_str, arg1, arg2, arg3, arg4, arg5, arg6 )

    char    *format_str;    /* printf format string         */
    int     arg1,           /* arg1 value                   */
            arg2,           /* arg2 value                   */
            arg3,           /* arg3 value                   */
            arg4,           /* arg4 value                   */
            arg5,           /* arg5 value                   */
            arg6,           /* arg6 value                   */
            level;          /* 1=normal mode, 3=debug mode  */

{
    FILE    *fp;
    char    dstr[10],
            tstr[10];
    time_t  clock;

    /*
     *  Ignore debug level messages if we are in normal operations mode.
     */

    if ( level > ErrorLogLevel )
        return;

    /*
     *  Open log file, if this doesn't work, return.
     */

    if ((fp = fopen(LogFile,"a")) == NULL)
        return;

    /*
     *  Add new line to log file, close the file.
     */

    clock = time( NULL );
    strftime( dstr, 9, "%D", localtime(&clock) );
    strftime( tstr, 9, "%T", localtime(&clock) );

    fprintf( fp, "%s %s - ", dstr, tstr );
    fprintf( fp, format_str, arg1, arg2, arg3, arg4, arg5, arg6 );
    fprintf( fp, "\n" );

    fclose( fp );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  error_handler()
*
* MODULE FUNCTION:
*
*   This routine is called when the GCB detects errors.  This routine is called
*   to display a message to the user, log an error to the logfile, and then
*   possibly exit, depending on the type of error.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  error_handler( error_num, module )

    char    *module;
    int     error_num;

{
    switch ( error_num )
    {
    case ERR_SYM_ATTRIBS :
    user_ack("Fatal error: can't get symbol attributes - exiting");
    elog(1,"%s: Fatal error: can't get symbol attributes - exiting",module);
    exit( ERR );
    break;

    case ERR_ADD_SYMBOL :
    user_ack("Fatal error: could not add Element name to symbol table - exiting");
    elog(1,"%s: Fatal: could not add Element name to symbol table",module);
    exit( ERR );
    break;

    case NO_ELEMENT :
    user_ack("Can't add symbol, invalid Element. Please select or create an Element");
    break;

    case LIB_ELEM_SYM :
    user_ack("Sorry but the GCB only allows a restricted set of symbols in Lib Elements")
;
    break;
    }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  get_filename_ptr()
*
* MODULE FUNCTION:
*
*   This routine extracts from a path name the name before the last /
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

char  *get_filename_ptr( filename )

    char *filename;

{
    char *ptr;

    ptr = filename;

    /*
     *  Find the end of the string.
     */

    while ( *ptr != NULL )
        ptr++;

    /*
     *  Move back one char at a time until either the
     *  start of the string is found, or a "/" is found.
     */

    while ( ptr != filename )
        if ( *ptr == '/' )
            return( ++ptr );
        else
            ptr--;

    return( ptr );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  get_sym_map_entry()
*
* MODULE FUNCTION:
*
*   This routine, given a widget, returns its Symbol_Map entry.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

Symbol *get_sym_map_entry( symbol )

    Widget  symbol;

{
    int     i;

    for ( i=0; i<MAX_SYMBOLS; i++ )
        if ( Symbol_Map[i].mycanvas == symbol )
            return( &Symbol_Map[i] );

    elog(1,"get_sym_map_entry: failed, returning 0");

    return( NULL );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  load_font()
*
* MODULE FUNCTION:
*
*   This routine performs the XLib calls to load the specified font.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  load_font( font, fnt_list )

    char        *font;
    XmFontList  *fnt_list;

{
    XFontStruct *font_struct = NULL;

    font_struct = XLoadQueryFont( display, font );
    if (! font_struct)
        {
        user_ack( "Couldn't XLoad font");
        return( ERR );
        }

    *fnt_list = XmFontListCreate( font_struct, XmSTRING_DEFAULT_CHARSET );

    return( 0 );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  load_help_file()
*
* MODULE FUNCTION:
*
*   This routine searches the help file for the parameter string.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

void  load_help_file( str )

    char *str;

{
    char cmd[ 256 ];

    /*
     *  Build a temporary file containing the help text for the current token.
     */

    strcpy( cmd, "more +/\042*" );
    strcat( cmd, str );
    strcat( cmd, "\042 " );
    strcat( cmd, Swd );
    strcat( cmd, "/GCBDoc" );
    strcat( cmd, " > /tmp/gcb.tmp 2>/tmp/gcb.err" );

    if ( system( cmd ) == ERR )
        {
        user_ack("Error reading documentation file, help text is not available");
        return;
        }
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  openFile()
*
* MODULE FUNCTION:
*
*   This routine makes a Unix call to open the file with the specified mode.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

FILE  *openFile( file, option)

    char    *file,
            *option;

{
    FILE    *filePtr;
    char    explain[250],
            tmpString[22];

    if (! (filePtr = fopen( file, option)))
        {
        switch ( option[0] )
            {
            case 'r' :  strcpy( tmpString, "reading" );
                        break;
            case 'a' :  strcpy( tmpString, "appending" );
                        break;
            case 'w' :  strcpy( tmpString, "writing" );
                        break;
            default  :  strcpy( tmpString, "bad open file option" );
            }
        sprintf( explain, "Can not open %s for %s", file, tmpString );
        user_ack( explain);
        return( NULL );
        }
    else
        return( filePtr );
}
```

```
/*****************************************<----->*****************************************
*
* MODULE NAME:  open_read_defaults()
*
* MODULE FUNCTION:
*
*   This routine reads from user's defaults file.
*
*
* REVISION HISTORY:
*
*   Graphical Comp Builder - MOTIF Release 1.0  - 07/17/91
*                            Release 1.01 - 08/01/91
*                            Release 1.02 - 08/28/91
*
*****************************************<----->*****************************************/

int  open_read_defaults()
{
    FILE    *fp;
    int     rc;
    char    option[132],
            value[132],
            value_str[132];

    /*
     *  Open the defaults file.
     */

    if (! (fp = fopen(DefaultsFile,"r")) )
        return( NOT_FOUND );

    /*
     *  Loop through the options.  Set the global vars used by the
     *  GCB to the value in the defaults file.
     */

    while (fscanf(fp,"%s%[^\n]",option,value_str) != EOF )
        {

        /*
         *  Throw out comment lines and lines with no value for the
         *  specified option.  Take the raw value string, which may
         *  contain leading spaces and tabs, and extract just the
         *  good stuff.
         */

        value[0] = '\0';
        rc = sscanf( value_str, "%s", value );

        if ((option[0] == '#') || (rc < 1))
            continue;

        /*
         *  We have a valid option and a value to go with it, determine
         *  which option and set flags accordingly.
         */

        else
            {
            if (! strcmp(option,"LIBRARY_PATH") )
                strcpy( LibPath, value );
```

```
if (! strcmp(option,"DISPLAY_FILE") )
    strcpy( DisplayFile, value );

if (! strcmp(option,"COMP_FILE") )
    {
    strcpy( CompFile,  value );
    strcpy( CompDir,   value );
    strcat( CompDir,   ".DIR" );
    strcpy( GCompFile, value );
    strcat( GCompFile, CMP_EXT);
    }

if (! strcmp(option,"ELEMENT_TYPE") )
    if (! strcmp(value,"LIB") )
        ElementType = LIB;
    else
        ElementType = ELEMENT;

if (! strcmp(option,"ELEMENT_FILE") )
    {
    if (ElementType == LIB)
        {
        strcpy( GElementFile, LibPath );
        strcat( GElementFile, "/");
        strcat( GElementFile, value );
        strcat( GElementFile, EL_EXT );
        }
    else
        {
        strcpy( GElementFile, value );
        strcat( GElementFile, EL_EXT );
        }
    strcpy( ElementFile, value );
    }

 if (! strcmp(option,"LOG_FILE") )
    strcpy( LogFile, value );

 if (! strcmp(option,"LOG_LEVEL") )
    ErrorLogLevel = atoi( value );

if (! strcmp(option,"MACROS_PATH") )
    strcpy( MacrosPath, value );

if (! strcmp(option,"OBJECT_TABLE") )
    strcpy( MSIDTable, value );

if (! strcmp(option,"POSITION") )
    strcpy( pPosition, value );

if (! strcmp(option,"POSITION_PATH") )
    strcpy( PositionPath, value );

if (! strcmp(option,"TARGET_LANG") )
    if ( strcmp(value,"C") == 0 )
        TargetLanguage = C;
    else if ( strcmp(value,"MOAL") == 0 )
        TargetLanguage = MOAL;

if (! strcmp(option,"USER_FUNCS_PATH") )
    strcpy( UserFuncsPath, value );

if (! strcmp(option,"WS_GLOBALS") )
    strcpy( WSGlobals, value );

        if (! strcmp(option,"DISPLAY_TOGGLE") )
            LogOrCompText = atoi( value );

        /*
         *  Throw out any unknown options.
         */

        else
            ;
        }
    }

return( OK );
}
```